

СОДЕРЖАНИЕ

Введение	3
1 Аналитический обзор существующих аналогов.....	5
2 Проектирование программного средства.....	11
2.1 Структура программы.....	11
2.2 Интерфейс программного средства.....	11
2.3 Проектирование функционала программного средства.....	13
3 Разработка программного средства.....	18
3.1 Создание контент-сущностей для интернет – магазина.....	18
3.2 Обработка заказов в интернет – магазине.....	18
3.3 Приём платежей.....	19
3.4 Доставка товаров.....	20
4 Тестирование программного средства.....	21
4.1 Отладочные процессы.....	21
4.2 Тестирование программы.....	21
4.3 Результаты тестирования.....	21
5 Руководство пользователя.....	22
Заключение	24
Список использованных источников	25
Приложение А. Исходный код программы	26

ВВЕДЕНИЕ

Для полномасштабной реализации проекта необходимо чётко расставить приоритеты: разобраться, что представляет из себя предметная область, определиться со структурой программы, запланировать ход действий и так далее.

Первый возникающий вопрос, а что за предметная область будет рассматриваться в рамках курсового проектирования? Моя задача – написать книжный интернет-магазин. На сегодняшний день, популярных книжных интернет-магазинов не так уж и много, в голову ничего не приходит, даже если очень хорошо подумать. Однако, такие крупные торговые гиганты, как Amazon, Ozon, eBay сразу приходят на ум. Данные веб-сервисы являются некоего рода “Эталон” в сфере интернет-магазинов, хоть и специализируются на всём, что попадает к ним в руки. На мой субъективный взгляд, такой подход не очень хорош, так как такого рода веб приложения созданы с целью заработать на пользователях ресурса, требуют огромного количества тестов, контроля, требуют пристального внимания со стороны разработчиков. С другой стороны, чем больше охват – тем больше посетителей будут обращаться с целью приобрести тот или иной товар, так как наличие обширного выбора товаров на любой вкус и цвет гарантирует большой приток новых пользователей. Хотя, если расставлять приоритеты при создании, конечно, чем больше охват – тем лучше, тем больше денег смогут заработать не только разработчики данного ресурса, но и заказчик, так как такого рода продукты, если они не теряются среди уже существующих, очень быстро окупаются и приобретают всеобщий интерес. С виду может показаться, что написание своего интернет-магазина – простая и не трудоёмкая задача. Однако, за такой простой и понятной внешней реализацией (frontend) скрывается куча проверок (backend), сложная логика отправки и принятия HTTP запросов, огромная клиентская база, с которой так или иначе тоже приходится взаимодействовать. Конечно, написать аналог интернет-гиганта у меня не получится, так как эти продукты писались группой разработчиков ни один год, однако я постараюсь сфокусироваться на одной лишь, книжной области, и создать приложение, которое сможет обслуживать клиентов, через которое можно будет собирать заказ из нужных книг, приобретать их, оформлять доставку и другие возможности, которые будут добавляться в ходе написания приложения.

Далее, необходимо принять решение, какова будет структура моего будущего веб-приложения. В первую очередь, так как я нацелен создать хорошо структурированный, рабочий и, самое главное, востребованный веб-продукт, я планирую внедрить в свой курсовой проект и реализовать такие технологии, методологии и паттерны, как: Работа с MS SQL, git и GitHub, Agile methodology (Backlog, Users annotations, Iterations), Modules testing, Design patterns, SOLID principles, DDD (Domain Driven Design), MVC и Entity Framework.

Последнее, что осталось зафиксировать перед стартом написания самой программы: необходимо определиться, какие проблемы (issues) будут решаться по ходу проектирования. Таким образом, постепенно, будут решаться одна задача за другой, что в конечном итоге, приведёт к полноценному продукту. Такой подход является главным принципом Agile methodology, так как действуя по такому плану, каждое маленькое действие будет внедряться и проверяться, и нахождение ошибок будет заметно проще, по сравнению с тем, если бы я сразу писал весь код, а дальше ломал голову, почему она не запускается. Исходя из анализа предметной области, я постарался выявить несколько самых главных задач, которые необходимо будет решить в самую первую очередь. Все эти задачи я зафиксировал в GitHub. Ниже приведены некоторые из них (p.s. Конечно, описание всех задач и проблем сразу не приведёт нас к результату, так как хорошей практикой является постепенное внедрение всех проблем, одной за другой. Какие-то проблемы могут быть плохо решаемыми, какие-то, наоборот, невозможными в реализации, ну или просто, сложно реализуемыми):

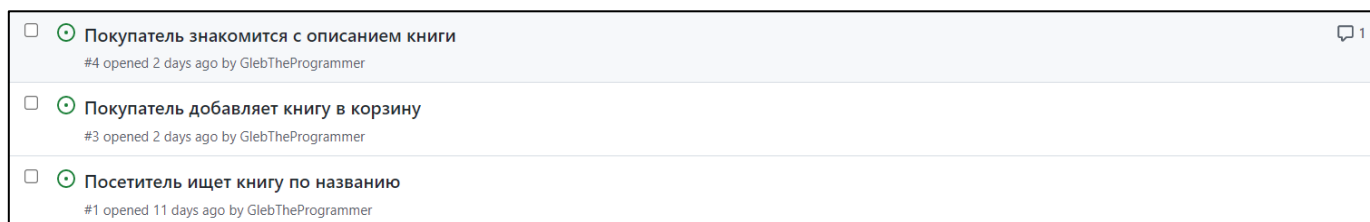


Рисунок 0.1 - Некоторые из начальных issues

Таким образом, подытоживая, создание полномасштабного продукта – сложная и практически невыполнимая задача для одного разработчика. Однако, в рамках своего курсового проектирования, я не буду гнаться за чем-то новым, не буду думать, а как мне реализовать ту или иную фишку, которая используется в одном из интернет-гигантов. Я постараюсь реализовать основные функции, постараюсь внедрить все технологии, которые были указаны выше, а также написать хорошо структурированную программу, вносить изменения в которую и редактирование уже существующих возможностей не составит труда.

1 АНАЛИТИЧЕСКИЙ ОБЗОР СУЩЕСТВУЮЩИХ АНАЛОГОВ

И так, как было указано выше, одними из самых главных продуктов, на которые все ссылаются при создании своего собственного интернет-магазина, являются Amazon, Ozon и eBay. Ниже приведена краткая характеристика и анализ каждого из них.

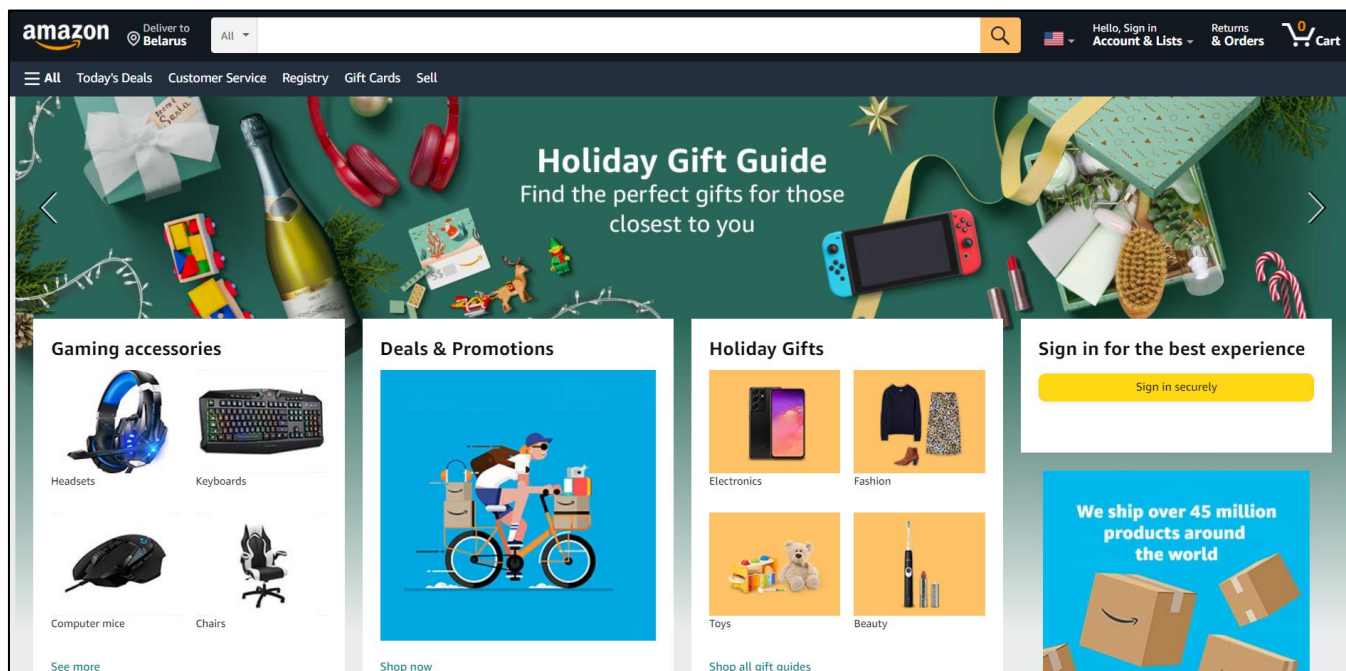


Рисунок 1.1 – Интернет-магазин “Amazon”

Перед нами – самый популярный интернет-магазин в мире. Её основатель Джефф Безос – один из самых богатейших людей в наше время. Огромная клиентская база Amazon говорит сама за себя. Только вдумайтесь, в компании работает 1 298 000 человек (на 2020 г.). Конечно, самим веб-сервисом занимается менее 1%, однако, такое количество сотрудников требует огромных затрат со стороны компании: поддержание хороших условий труда, выплата премий и надбавок за успешную работу, разные акции и ивенты для сотрудников компании. Этот продукт показывает, как за четверть века может вырасти компания, состоящая из одного руководителя и нескольких сотрудников до компании такого масштаба. (Если провести параллель с языками программирования, резкая популярность Amazon очень сильно напоминает рост популярности первого языка программирования - Fortran. Однако, на сегодняшний день, Fortran практически не используется, так как все нынешние языки программирования превосходят его, как и в скорости, так и в удобствах написания кода. Amazon же, в свою очередь, умудряется находиться на первых местах в своей области на протяжении долгого времени, что, скорее всего, продолжится ещё не один десяток лет). Оценивать недостатки такого рода продуктов, с моей стороны, не очень правильно и

корректно, поэтому я постараюсь выявить главные плюсы, которые я, в свою очередь, возможно, привнесу в свой проект.

Преимущества:

- Приятный, не выделяющийся интерфейс, где всё чётко структурировано и понятно;
- большой спектр выбора: множество категорий, по которым можно найти интересующий нас товар в кратчайшие сроки;
- существует возможность поменять язык сайта, которая помогает ориентироваться пользователям разных национальностей;
- присутствует корзина с товарами, которая позволяет добавлять в будущий заказ сразу несколько товаров, что помогает сэкономить на доставке;
- размещена кнопка с выпадающим меню, которое включает возможности навигации по сайту;
- наличие строки поиска, взаимодействуя с которой пользователь может найти именно тот товар, который ему нужен, если он, конечно же, существует.

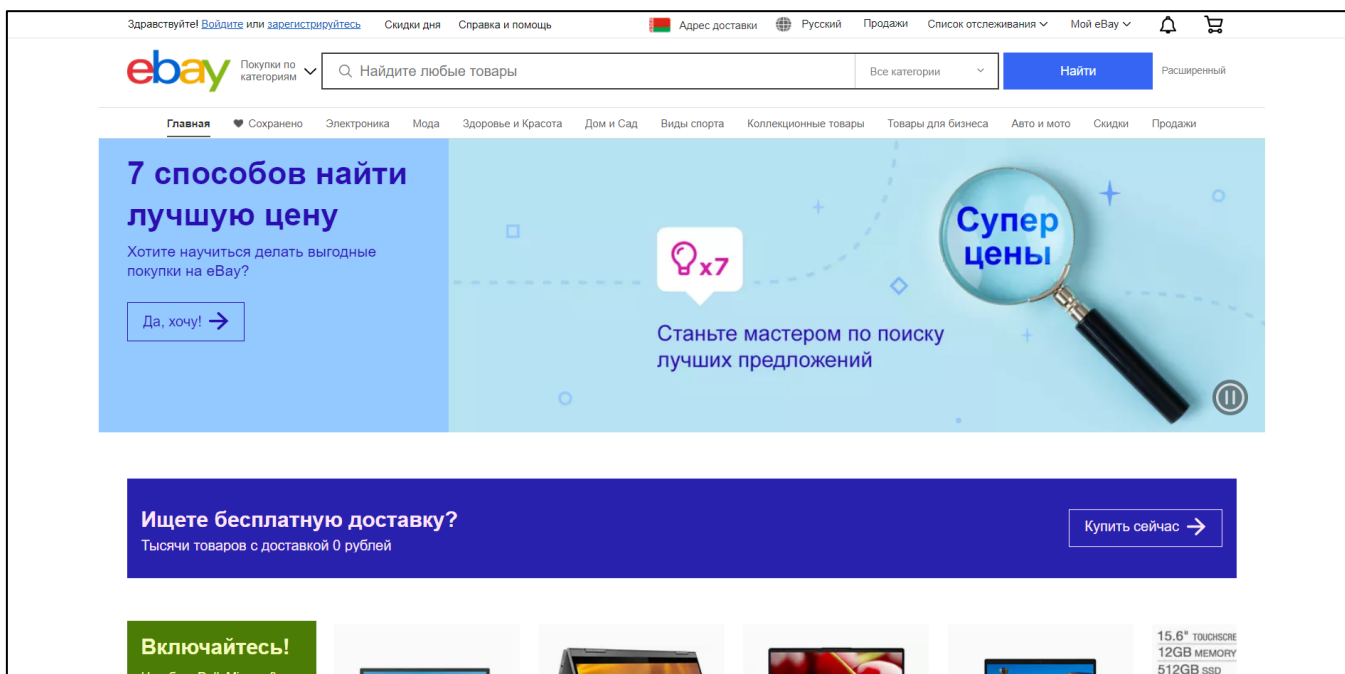


Рисунок 1.2 – Интернет-магазин «eBay»

На рисунке расположен скриншот сайта eBay. Конечно, по популярности, он очень сильно отстаёт от Amazon, однако это мешает ему находиться в топе среди других и занимать почётное место. Идея создания лежит ещё с 1995 года, когда у основателя Дэвида Уэнига возникла мысль создать свой уникальный продукт. В какой-то степени, у него это получилось: Количество сотрудников на декабрь 2018 года составляло около 14 000, а оборот компании исчисляется десятками миллиардов \$. Первые офисы возникли в США, что принесло популярность среди

англоговорящих, хотя на сегодняшний день, по собственному опыту, могу сказать, что этот интернет-магазин также является популярным среди русскоговорящего населения, так как большинство моих родственников заказывают товары именно из этого магазина. Сперва может показаться, что продукт создавался русскими разработчиками, так всё чётко и понятно, однако это не так. Чтобы сильно не повторяться, постараюсь выделить основные плюсы, не учитывая те, которые были приведены выше.

Преимущества:

- Наличие возможности настраивать уведомления на главной странице, что позволяет отслеживать интересующие нас товары и узнавать о их поступлении в кратчайшие сроки;
- на главной странице присутствуют провокационные баннеры, которые предлагают сэкономить, участвуя в акциях, которые открываются при переходе;
- на главной странице присутствуют провокационные баннеры, которые предлагают сэкономить, участвуя в акциях, которые открываются при переходе;
- удобное меню категорий на главной странице сайта.

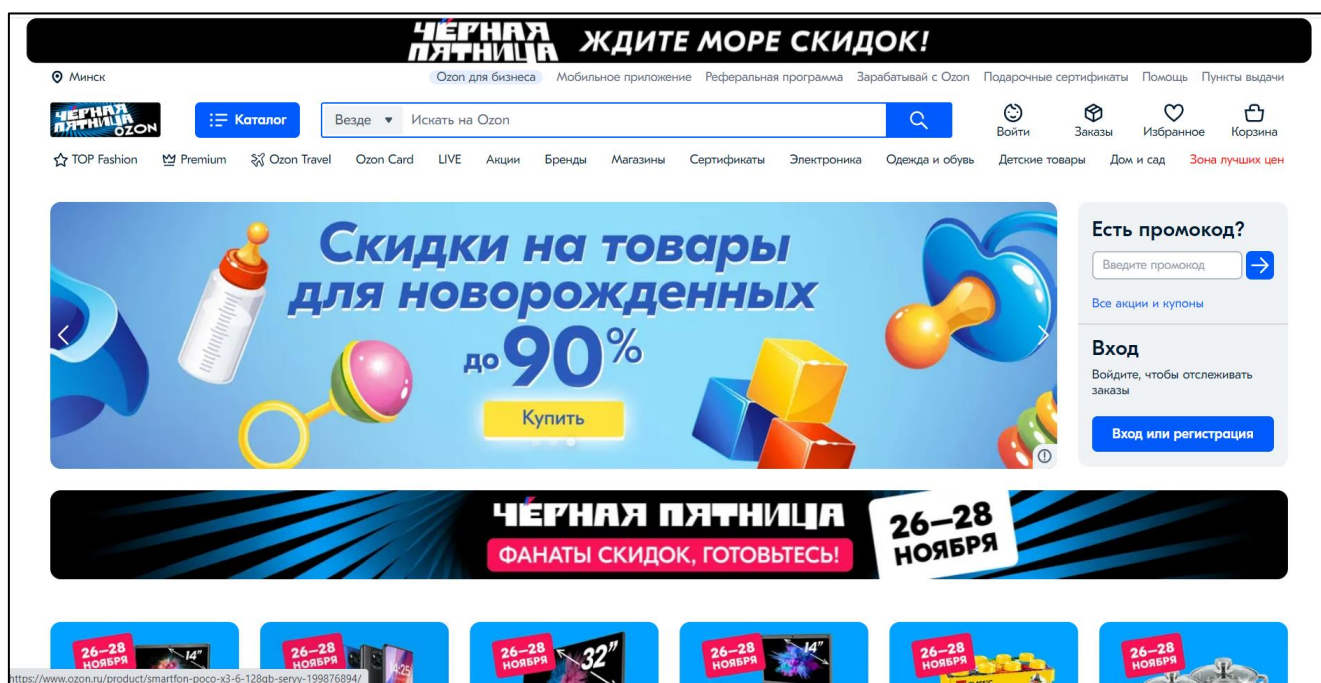


Рисунок 1.3 – Интернет-магазин «Ozon»

Последний магазин, который я планирую рассмотреть перед началом проектирования – магазин Ozon. Перед нами – исконно русский продукт, который недалеко ушёл от американского конкурента – eBay. Основатели Александр Егоров и Дмитрий Рудаков предложили идею о создании данного продукта в 1998 году. Магазин стал одним из самых популярных в России за пару лет. Такую популярность магазин приобрёл не только из-за того, что это был один из самых первых интернет-магазинов в СНГ, но и благодаря сотрудничеству с крупными брендами, взаимодействуя с которыми, производил доставку на их склады товаров различных категорий. Оборót сотрудников компании на начало 2020 года превышал 12 000 человек. Магазин пользуется спросом не только на родине, но и в близлежащих странах, так как поддерживает доставку за рубеж, как курьером, так и самолётом, по воздуху. В оформлении магазина, можно выделить следующие достоинства:

Преимущества:

- Наличие строки с возможностью ввести промо код, чтобы получить бонусы, скидки, возврат процента от покупки обратно на карту и так далее;
- возможность добавлять товары в категорию “избранные”, что позволяет вернуться к интересующим нас товарам в удобное для нас время;
- отображение акций и различных предложений, которые могут быть интересны обывателям и посетителям магазина.

Вывод: Рассмотрев аналоги, приведенные выше, можно подметить, что их функционал не сильно отличается, за исключением мелких, несущественных “изюминок”. Каждый интернет-магазин представляет собой локальную страницу, на которой представлен весь функционал сайта, а также множество полей и кнопок, которые образуют User Interface. Я же в своей курсовой работе постараюсь создать программу, которая будет похожа на все вышеперечисленные, которая будет включать в себя некоторые из преимуществ аналогов, в то же время, будут реализованы свои собственные идеи, которые должны будут сделать внешний вид и функционал магазина максимально приближенным к интернет-гигантам.

Постановка задачи:

Целью своего курсового проектирования я, в первую очередь, ставлю создание удобного, практичного и понятного пользователю приложения, которым он сможет пользоваться при необходимости заказать те или иные книги.

Темой моего курсового проектирования является создание программного приложения «Книжный интернет-магазин». Для выполнения работы в полном

объёме, необходимо составить чёткий, структурированный план, на который можно будет ориентироваться при написании программы.

В первую очередь, необходимо продумать графическую составляющую программы таким образом, чтобы она была понятна пользователю, в какой-то мере удовлетворяла нынешним стандартам и не содержала в себе однотипных элементов.

Программа будет представлять собой веб-страницу, на которой будет расположен основной функционал. В то же время, переход между страницами будет осуществляться не путём создания новых страниц (такой подход будет максимально неудобным для пользователя, так как придётся взаимодействовать со всеми страницами сразу, сворачивая ненужные и открывая те, на которых необходимо произвести дальнейшие манипуляции), а путём манипуляции с HTTP протоколом. Создавая всё новые и новые контроллеры (Controllers), можно будет обрабатывать те или иные действия пользователя на основной странице, подгружая всё новые и новые представления (Views), и переходить между ними при помощи самого HTTP. На самих представлениях будут отображаться модели (Models), взаимодействуя с которыми пользователь сможет исследовать весь функционал веб-приложения. Такой подход является типовым в программировании (MVC), которого придерживаются все самые популярные (и не только) компании, проектируя свои приложения.

При запуске программы, пользователю будет представлена основная веб-страница, на которой будет представлен материал с информацией о предметной области, а также немного информации о ходе написания работы.

Теперь немного о функционале. Пользователю будет дана возможность добавлять интересующие его товары в корзину, а также удалять в случае ошибки. После того, как пользователь добавит интересующие его позиции в корзину, ему необходимо будет оформить заказ. Оформление будет включать в себя: подтверждение номера телефона через код, который будет отправляться в случае предоставления пользователем корректного номера.

В программном средстве планируется реализовать следующие функции:

- Поиск книг по названию;
- поиск книги по ISBN номеру или автору;
- возможность полноценному покупателю добавлять книгу в корзину;
- добавить реализацию самой корзины для хранения товаров покупателя;
- добавить описание к каждой из книг;
- реализовать просмотр корзины в удобном формате;

- реализовать возможность оформления заказа – подтверждения мобильного телефона;
- возможность покупателю выбирать способ доставки;
- функцию просмотра информации о заказе после его оформления;
- отправка сотруднику письма с информацией о собранном заказе;
- подключить SQL EXPRESS DB для хранения информации о книгах и заказах;
- возможность покупателю оплачивать как наличными, так и банковской картой;
- функцию программисту получать информацию обо всех ошибках, полученных пользователями при взаимодействии с сайтом для их дальнейшего исправления.

Для разработки программного средства будут использоваться языки программирования C#, C++, среда разработки Visual Studio Community Edition 2019. Также, для реализации визуальной составляющей веб-страниц, будет использоваться HTML, CSS и JavaScript.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Структура программы

В данном программном средстве, код программы будет разбит на слои, призванные разграничить области разработки:

- application layer – Слой, который будет содержать непосредственно те объекты, с которыми будет взаимодействовать пользователь, фактически, этот слой можно назвать верхним слоем инкапсуляции, то есть это то, что будет видеть пользователем, не вдаваясь, как это хранится;
- domain layer – Слой, который будет содержать структуру объектов, с которыми предстоит взаимодействовать пользователю, те, которые будут передаваться в готовом виде в application layer. Является нижним слоем инкапсуляции, то есть содержит непосредственно поля, их обработку и реализацию;
- infrastructure layer – Слой, отвечающий за взаимодействие программы с базой данной SQL EXPRESS. Он содержит в себе все необходимые методы и настройки, которые позволяют производить трансфер в обе стороны;
- plugin layer – Слой, отвечающий за взаимодействие программы с внешними сервисами, например, Яндекс кассой, а также сервисом для отправки сообщений на номера телефонов и через электронную почту;
- presentation layer – Слой, отвечающий за представление, другими словами, слой будет содержать код, представляющий визуальный вид веб-страниц.

2.2 Интерфейс программного средства

Одним из наиболее важных критериев качества программного обеспечения является интуитивно-понятный интерфейс со всем необходимым функционалом, но без лишних деталей. В данном программном средстве будет использован классический стиль Windows, который, в отличие от многих других стилей, не привлекает к себе ненужное внимание, и, следовательно, не отвлекает.

Главная страница(рисунок 2.1) будет состоять из трёх компонентов:

- <<Панель поиска>>;
- <<Кнопка поиска>>;
- <<Надпись-ссылка о количестве товаров в корзине>>.

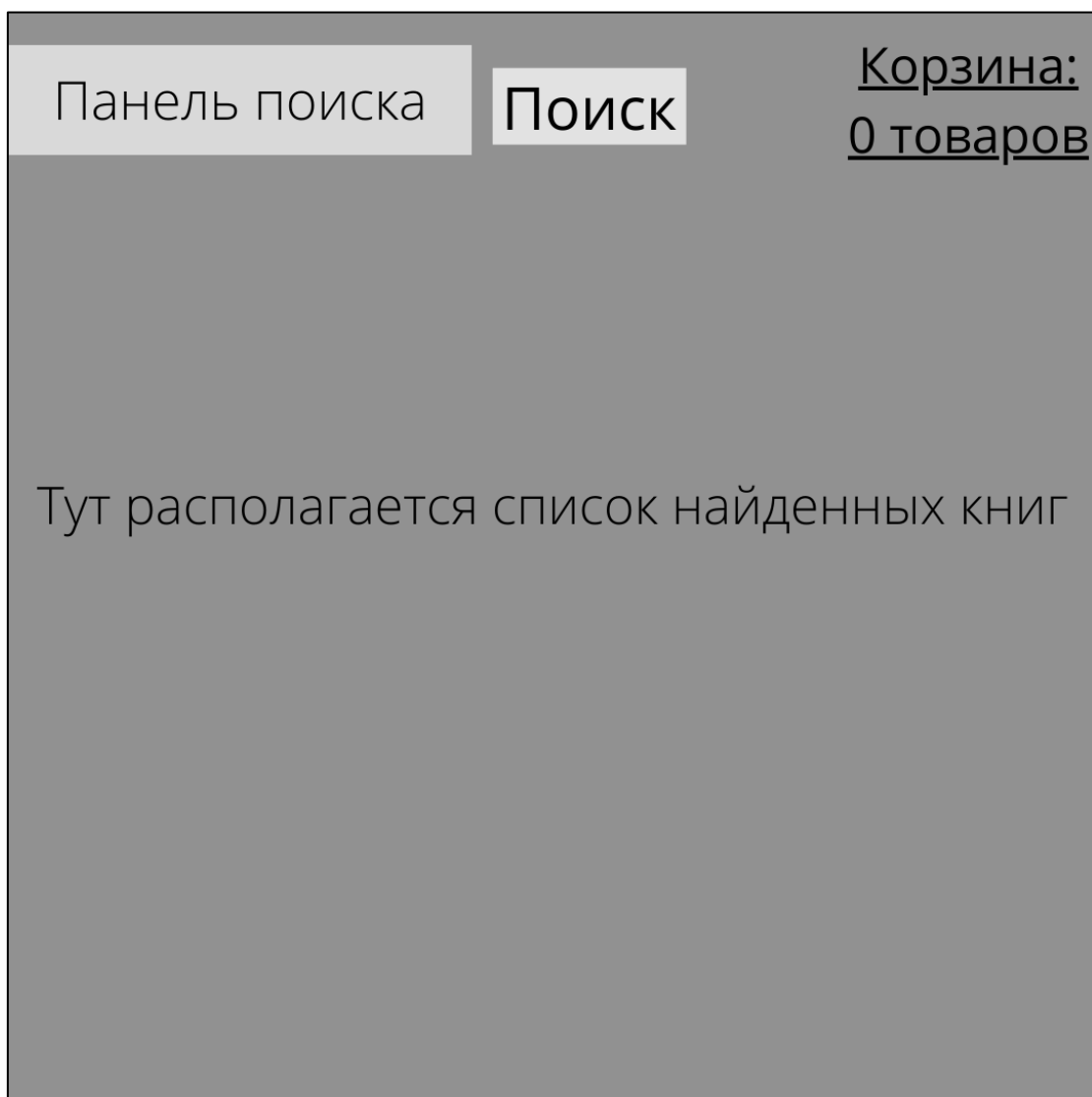


Рисунок 2.1 – Запланированный интерфейс главной страницы

При нажатии кнопки «Поиск» будет производиться поиск всех книг, которые будут содержать в себе любую информацию, которую пользователь должен будет предварительно ввести в «Панель поиска», будь то название, автор, или же ISBN номер книги. Все найденные программой книги будут располагаться в области под панелью поиска

При нажатии на надпись-ссылку «Корзина: N товаров» пользователь будет перенаправлен на веб-страницу с корзиной, и всеми добавленными в неё товарами. Интерфейс корзины изображён на рисунке 2.2.

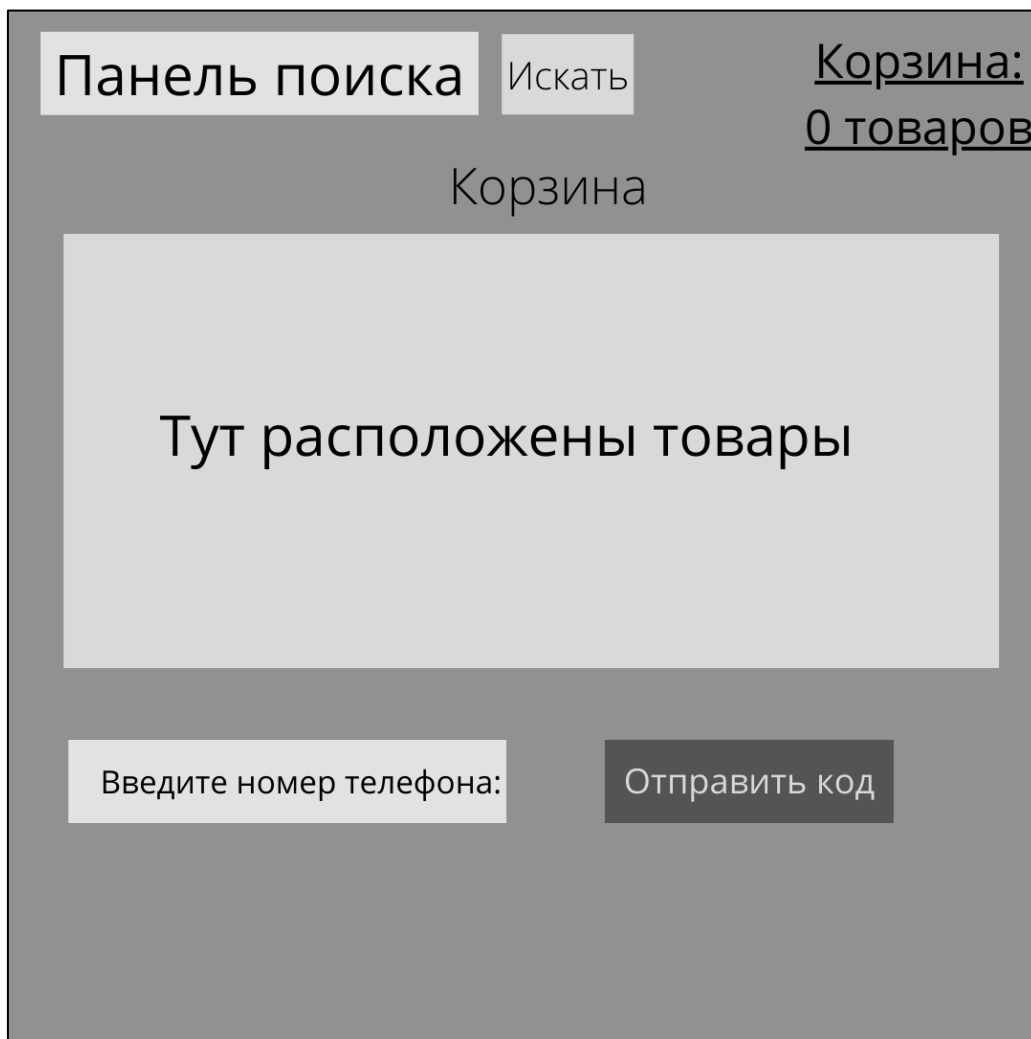


Рисунок 2.2 – Запланированный внешний вид веб-страницы с корзиной

Панель поиска никуда не пропадает, на случай, если пользователь решит дальше искать и добавлять книги в корзину, однако появляется новая панель для ввода номера телефона пользователя для оформления заказа, а также кнопка для отправки SMS-оповещения с кодом, который пользователю необходимо будет ввести на следующей странице.

Остальные же страницы будут представлять из себя меню с выбором доставки, способа оплаты, а также возможность повторно просмотреть корзину.

2.3 Проектирование функционала программного средства

Грамотно поставленная задача и хорошо составленные алгоритмы – ключевая фаза в проектировании программного средства. Программное средство «Книжный интернет-магазин» должно предоставлять пользователю такой минимальный функционал как:

- Поиск книг по автору или названию, а также по ISBN-номеру;
- добавление книги в уже существующую корзину, либо же в новую;
- удаление уже добавленных товаров из корзины;

- возможность оформить доставку и произвести оплату.

2.3.1 Поиск книг по автору, названию или ISBN-номеру

Поиск необходимых книг будет происходить путём обращения программы к базе данных существующих книг с использованием полномасштабного поиска, который будет работать как с названием, так и с ISBN номером и авторами книг.

Далее, найденные книги программа будет загружать в программу для отображения, после чего будет вызвано соответствующее представление, которое будет призвано отобразить все найденные программой книг.

Блок-схемы по поиску книг приведены на рисунке 2.3.

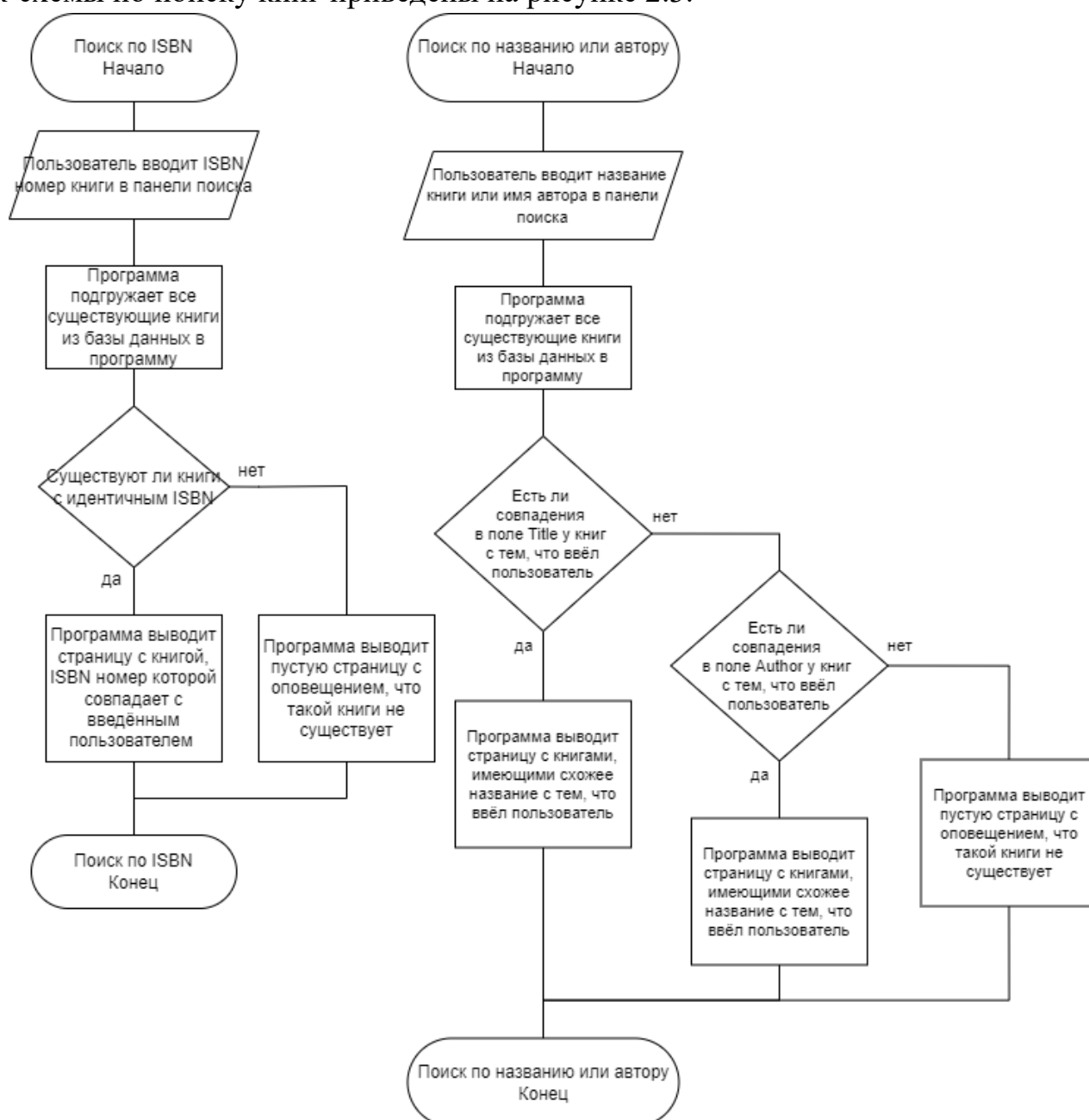


Рисунок 2.3 – Блок-схемы процедур по поиску книги

2.3.2 Добавление книг в корзину

Добавление книг в корзину будет происходить путём обращения к базе данных. Сначала, будет осуществлён поиск уже существующей корзины с идентичным идентификатором как у сессии. Если корзина ещё не была создана – будет создана пустая корзина, которая будет загружена программе на обработку. Программа же вызовет соответствующее представление для отображения.

Блок-схема по добавлению книги в корзину приведена на рисунке 2.4.

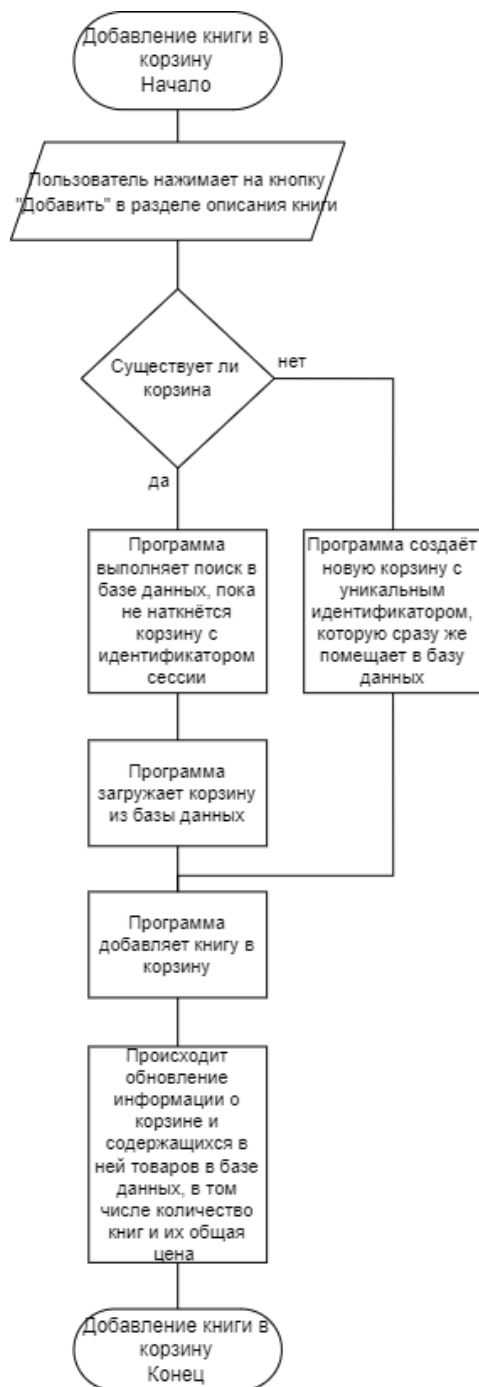


Рисунок 2.4 – Блок-схема процедуры по добавлению книги в корзину

2.3.3 Удаление товаров из корзины

Удаление товара из корзины, в первую очередь, будет означать удаление товара из базы данных в первую очередь, после чего будет произведена обычная отрисовка новой корзины, путём вызова представления.

Блок-схема по удалению товаров из корзины приведена на рисунке 2.5

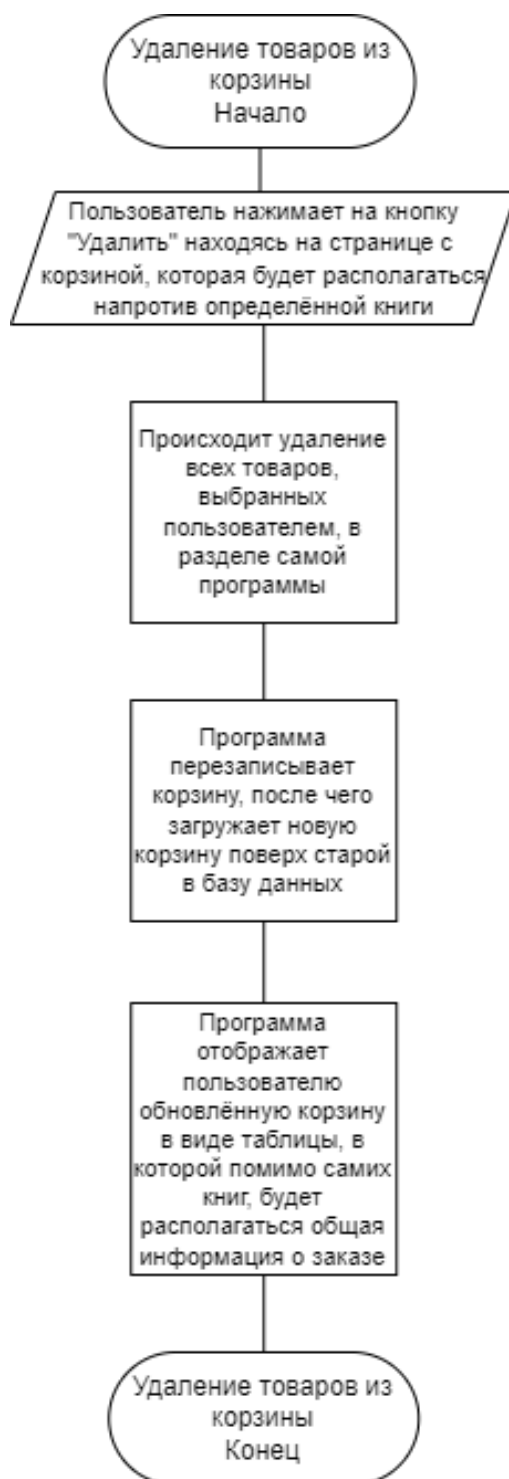


Рисунок 2.5 – Блок-схема процедуры по удалению товаров из корзины

2.3.4 Выбор способа доставки и оплата

После того, как корзина будет собрана, пользователь сможет приступить к оформлению доставки путём выбора мест на карте, либо же в выпадающем меню.

После выбора места доставки, пользователь сможет выбрать удобный для себя способ оплаты (например, через банковскую карту). Как только оплата будет совершена, процедуру заказа можно считать выполненной.

Блок-схемы по выбору способа доставки и оплате приведены на рисунке 2.6

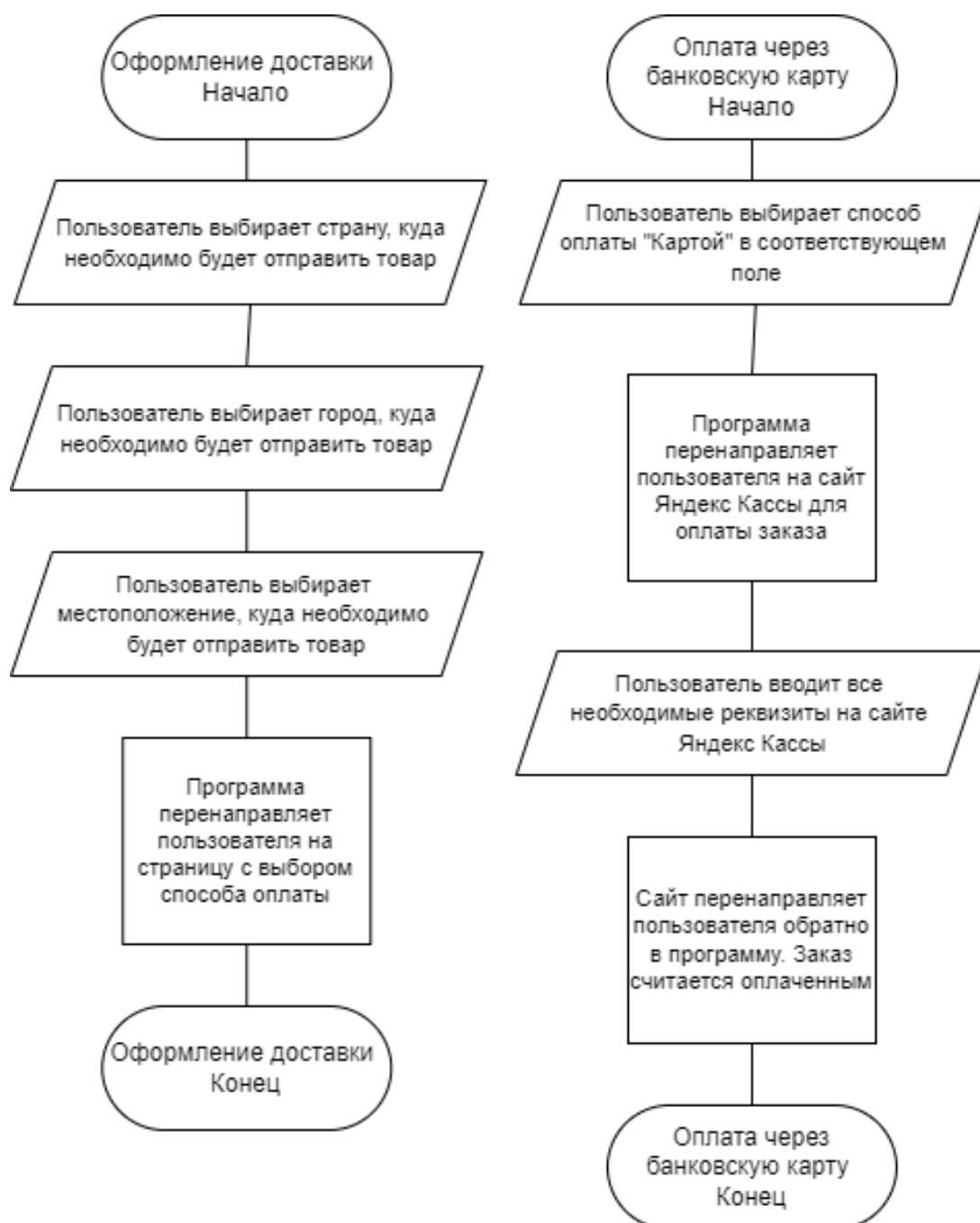


Рисунок 2.6 – Блок-схемы по выбору способа доставки и оплаты

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Создание контент-сущностей для интернет-магазина

Сайт – это витрина вашего интернет-магазина. И это именно то место, где покупатель принимает решение: купить у вас или пойти дальше. Поэтому между привлечением покупателей и оформлением заказа решающую роль в воронке продаж играет контент. Удобная и понятная подача контента с ответами на предположительные вопросы и сомнения покупателей и превращает случайного пользователя в вашего клиента.

Поэтому очень важно, чтобы контент был:

1. Полезным для покупателей;
2. уникальным для поисковых систем.

Контент в интернет-магазине бывает разным, и чем больше его форм представлено на сайте, тем лучше. Это могут быть тексты, фотографии, инфографики, 3d-обзоры, видеоролики, диаграммы – все, что поможет убедить купить ваш товар. В моём интернет-магазине, основной контент-сущностью будут являться книги, а набор книг будет являться заказом.

Текстовая информация обязательно должна присутствовать на сайте, так как и покупатели, и пользователи в первую очередь оценивают именно по текстам, достоин ли сайт внимания или нет. Технические характеристики должны соответствовать инструкции – это единственная информация, которая может быть неуникальной. А вот описания и обзоры должны показывать, чем товар может решить проблемы покупателя и упростить ему жизнь. Возможность добавлять вопросы и отзывы о товарах позволит вашему сайту получать бесплатный уникальный контент от пользователей интернет-магазина. В моём программном средстве, графа описания книги не будет исключением. У каждой книги будет своё собственное описание. Если же рассматривать программную реализацию, то книга будет представлять из себя объект со своими полями, которые будут содержать название, имя автора, описание, цену и ISBN номер.

3.2 Обработка заказов в интернет-магазине

В контактном центре интернет-магазинов принимаются заказы по телефону и электронной почте через корзину на сайте. В моей программе это не будет исключением.

Кто бы ни обрабатывал заказы интернет-магазина, он должен иметь скрипты продаж – готовые диалоги, направленные на перехват инициативы, консультацию по товарам и увеличение количества позиций в чеке.

Обработка заказа включает следующие шаги:

1. Проверка складских остатков;
2. резервирование товара в базе данных;
3. согласование способа оплаты с пользователем;
4. заполнение адреса и времени доставки;
5. создание задачи по комплектации товара для службы доставки.

Очень важно, чтобы любое изменение по состоянию заказа после его обработки фиксировалось. Таким образом любой менеджер сможет отследить, на каком этапе находится заказ и ответить на все вопросы клиента.

Со временем многие магазины автоматизируют процесс обработки заказа за счет личного кабинета покупателя, где сам покупатель добавляет товары в корзину, выбирает способ оплаты, адрес и время доставки, и в зависимости от этого либо сам приходит за товаром в пункт самовывоза, либо курьер перезванивает покупателю за час до времени доставки и напоминает о своем прибытии. В своём интернет-магазине я планирую реализовать именно такой подход работы с клиентской базой.

3.3 Приём платежей

Чем больше на сайте будет вариантов оплаты, тем выше будет конверсия пользователей в покупателей. Согласно исследованию, проведенному торговым агрегатором Яндекс.Маркет и компанией GFK в 2015 году, покупатели отдают предпочтение оплате наличными при получении.

1. Наличные при получении – 71%.
2. Оплата онлайн банковской картой – 54%.
3. Оплата электронными деньгами – 31%.
4. Оплата картой в момент доставки – 34%.
5. Предоплата – банковский перевод – 16%.
6. Предоплата через терминал – 7%.

Как видно из процентовки, чаще всего покупатели желают оплатить товар банковской картой онлайн или курьеру при получении. Сайт необходимо интегрировать с платежными системами, которые позволяют принимать платежи

онлайн. Преимущество предоплаты онлайн в том, что вы избавляетесь от проблемы заморозки средств на товары, которые уже отправлены на доставку, но оплата за них еще не получена. При большом обороте набегает круглая сумма, которую можно было бы пустить на рекламу, закупку, аренду или развитие интернет-магазина. В этом плане моё веб-приложение не будет исключением.

3.4 Доставка товаров

Выбор вариантов доставки для интернет-магазина – это непростое дело, но важно одно: даже на старте должно быть представлено несколько видов доставки, чтобы у покупателей был выбор.

В Беларуси распространены следующие варианты доставки:

1. «БелПочта»;
2. Курьерская служба;
3. Собственная служба доставки;
4. Транспортная компания;
5. Пункт самовывоза.

По всему миру, в том числе в России и Украине активно используются почтоматы (постаматы) – ячейки для хранения и выдачи посылок. А в Британии, которая является мировым лидером на рынке электронной коммерции, этот вариант доставки предпочитают 80% британцев. Но в Беларуси эта услуга пока недоступна. Я же буду делать приложение, которое будет охватывать не только Беларусь, но и другие страны, поэтому, всё же, доставку будет удобнее всего реализовать через постаматы.

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

4.1 Отладочные процессы

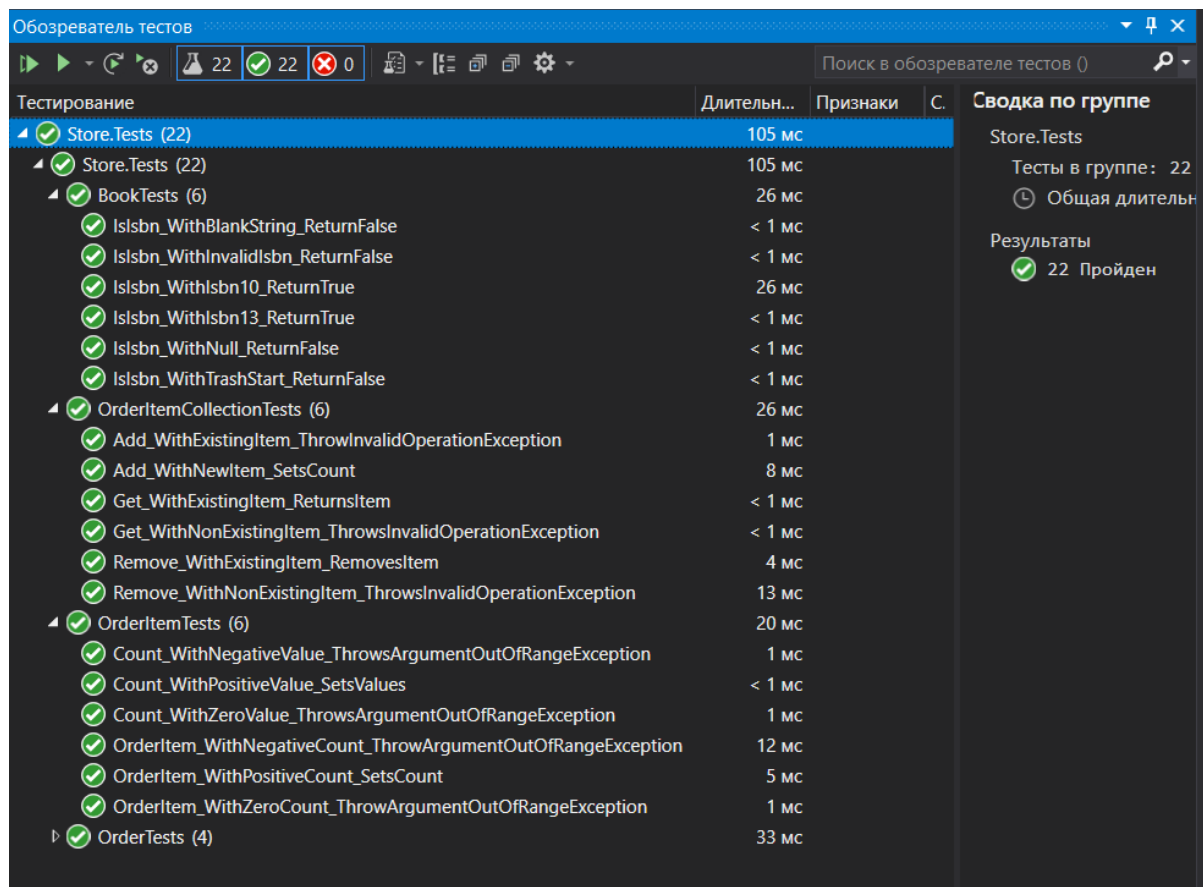
Для тестирования программного средства был задействован отладчик среды программирования Visual Studio Community Edition 2019, в которой непосредственно разрабатывалась программа, а также обратная связь со стороны первых пользователей и тестировщиков программы.

4.2 Тестирование программы

Для тестирования программного средства были созданы так называемые Unit-тесты, которые позволяют создавать множество виртуальных контент-сущностей, которые помогают удостовериться, что сайт функционирует нормально, без каких-либо происшествий.

4.3 Результаты тестирования

В ходе тестирования приложения не было выявлено никаких ошибок в работе сайта, о чём может свидетельствовать рисунок 4.1 с результатами работы Unit-тестов.



Тестирование	Длительн...	Признаки	С.	Сводка по группе
Store.Tests (22)	105 мс			Store.Tests
Store.Tests (22)	105 мс			Тесты в группе: 22
BookTests (6)	26 мс			Общая длительн
IsIsbn_WithBlankString_ReturnFalse	< 1 мс			Результаты
IsIsbn_WithInvalidIsbn_ReturnFalse	< 1 мс			22 Пройден
IsIsbn_WithIsbn10_ReturnTrue	26 мс			
IsIsbn_WithIsbn13_ReturnTrue	< 1 мс			
IsIsbn_WithNull_ReturnFalse	< 1 мс			
IsIsbn_WithTrashStart_ReturnFalse	< 1 мс			
OrderItemCollectionTests (6)	26 мс			
Add_WithExistingItem_ThrowInvalidOperationException	1 мс			
Add_WithNewItem_SetsCount	8 мс			
Get_WithExistingItem_ReturnsItem	< 1 мс			
Get_WithNonExistingItem_ThrowsInvalidOperationException	< 1 мс			
Remove_WithExistingItem_RemovesItem	4 мс			
Remove_WithNonExistingItem_ThrowsInvalidOperationException	13 мс			
OrderItemTests (6)	20 мс			
Count_WithNegativeValue_ThrowsArgumentOutOfRangeException	1 мс			
Count_WithPositiveValue_SetsValues	< 1 мс			
Count_WithZeroValue_ThrowsArgumentOutOfRangeException	1 мс			
OrderItem_WithNegativeCount_ThrowArgumentOutOfRangeException	12 мс			
OrderItem_WithPositiveCount_SetsCount	5 мс			
OrderItem_WithZeroCount_ThrowArgumentOutOfRangeException	1 мс			
OrderTests (4)	33 мс			

Рисунок 4.1 – Результат работы Unit-тестов программы

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска программы необходимо запустить файл Store.exe. Основная страница, появляющаяся при запуске программы, изображена на рисунке 5.1.

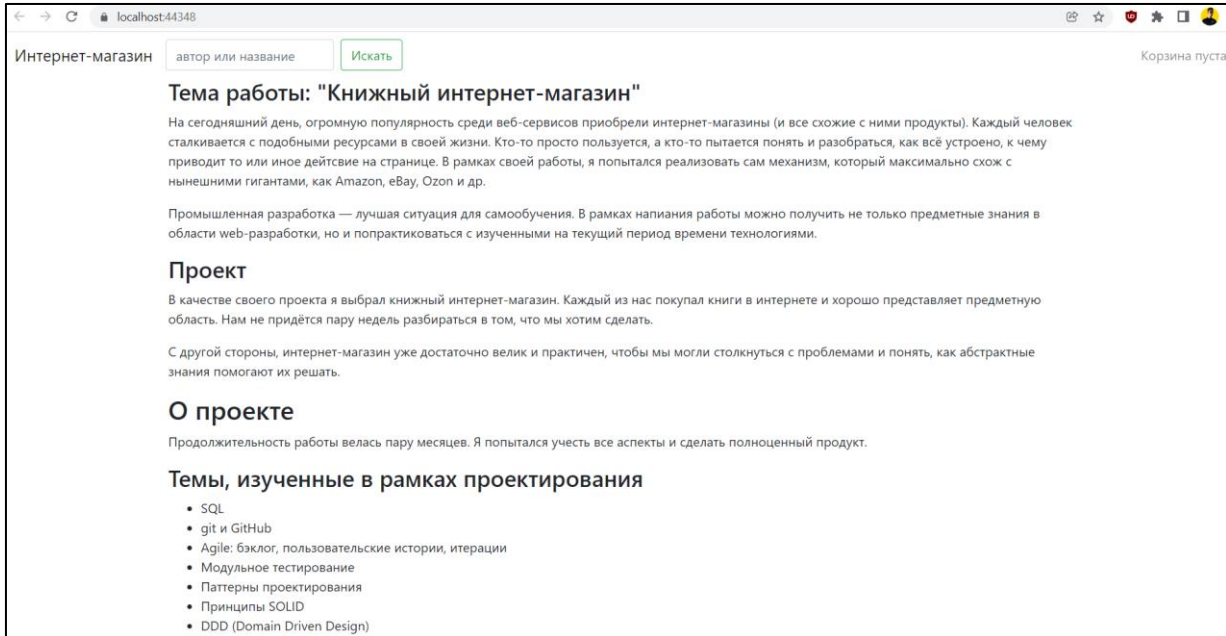


Рисунок 5.1 – Основная веб-страница интернет-магазина

Поиск книг осуществляется в соответствующем поле для ввода любых данных книги. Страница, показывающая результат поиска по слову “Programming”, изображена на рисунке 5.2.



Рисунок 5.2 – Результаты поиска

После того, как пользователь нажмёт на одну из книг, он будет перенаправлен на веб-страницу с описанием этой книги. Страница с описанием книги изображена на рисунке 5.3.



Рисунок 5.3 – Страница с описанием книги

Пользователю предоставляется возможность добавить книгу в корзину, после добавления нескольких книг в корзину, пользователь сможет перейти в неё путём нажатия на надпись-ссылку, расположенную справа на основной странице. Внешний вид корзины с товарами изображен на рисунке 5.4.

Интернет-магазин автор или название Искать Товаров 3 на сумму 34,6200

Корзина

№	Автор	Название	Количество	Цена	
1	B. W. Kernighan, D. M. Ritchie	C Programming Language	1	14,9800	Удалить
2	D. Knuth	Art Of Programming, Vol. 1	1	7,1900	Удалить
3	M. Fowler	Refactoring	1	12,4500	Удалить
Итого			3	34,6200	

Для оформления заказа необходимо подтвердить номер вашего мобильного. Введите номер и нажмите *Отправить код*. На ваш номер будет отправлен *код подтверждения*.

Мобильный телефон

+79876543210

Отправить код

Рисунок 5.4 – Страница со сформированной корзиной

Как только корзина будет собрана, и пользователь решит перейти к следующему шагу, ему необходимо будет ввести свой номер телефона, и нажать на кнопку “Отправить код”, после чего появится окно, показанное на рисунке 5.5.

Интернет-магазин автор или название Искать Товаров 3 на сумму 34,6200

Подтверждение мобильного

Введите код подтверждения из SMS.

Код подтверждения

0000

Проверить

Если на ваш номер SMS не пришла в течение минуты, проверьте правильность номера и повторите отправку кода.

Мобильный телефон

+7 987 654-32-10

Повторить код

Рисунок 5.5 – Страница с подтверждением мобильного телефона

Далее, будет произведено заполнение данных о доставке, способе оплаты на сторонних ресурсах, а также повторный просмотр корзины в целях убедиться, что все товары были собраны правильно, и что заказ не имеет в себе никаких лишних позиций.

ЗАКЛЮЧЕНИЕ

Актуальность данного веб-сервиса очевидна. Это обусловлено тем, что мир постепенно начинает переходить в виртуальный формат, где удобство является одним из главных критериев. В нынешнее время книги начинают терять свою значимость, тем не менее, большая часть населения заинтересована в собственном развитии и самообразовании. Книга становится нашим другом с раннего возраста и сопровождает нас на протяжении всей нашей жизни. Помимо того, что книга является источником мудрости, знаний и науки, она является лекарством для души, позволяет узнать мир и способствует формированию мировоззрения. Следует помнить о том, что ни один гаджет, ни один фильм не передаст тех эмоций, чувств радости и волнения, которые испытываешь при чтении книги. Никакая компьютерная графика не сможет воплотить те представления о красоте нашей природы, которые описывают авторы произведений в своих рассказах и поэмах. Ни один фильм, даже если его сюжет написан по книге, не раскроет до конца характер главных героев, не сможет передать задумку автора произведения, не сможет передать чувства, которые переполняли его при написании. На протяжении многих лет книги являются одним из самых важных источников получения информации. Ранее не было такого понятия как «интернет», и потому при наличии кого-либо вопроса, ответ искали именно в книге. Книга является тем незаменимым инструментом, который учит нас жизни. Книги никогда не стареют, не теряют своего смысла и ценности. Главное – уметь выбирать настоящие книги: те, что действительно украшают нашу жизнь, делают её богаче и содержательнее, те книги, которые являются настоящим сокровищем человеческой мысли и опыта.

Целью данного курсового проекта являлось разработка полноценного программного средства, которое будет реализовывать основной функционал нынешних книжных интернет-магазинов. Задача была выполнена успешно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Visual Studio Product Documentation – VS Technologies [Электронный ресурс]. – Режим доступа:
http://docs.VisualStudio.com/products/Visual_Studio_community_2019.
- [2] Wikipedia [Электронный ресурс]. – Свободная энциклопедия. – Режим доступа:
https://ru.wikipedia.org/wiki/Book_Shop.
- [3] Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003.
- [4] Программирование на языке Delphi / Д. А. Сурков [и др.]. – учеб. пособие. – Режим доступа: <http://www.rsdn.ru/?summary/3165.xml>, – 2005.
- [5] Уилсон, С. Принципы проектирования и разработки программного обеспечения, учебн. курс. – СПб, 2003.
- [6] Шупрута, В. В. Delphi 2005. Учимся программировать: / В. В. Шупрута. – Москва: изд. «НТ Пресс», 2001. – 140 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программы:

Book Model

```
public class BookModel
{
    public int Id { get; set; }

    public string Isbn { get; set; }

    public string Author { get; set; }

    public string Title { get; set; }

    public string Description { get; set; }

    public decimal Price { get; set; }
}
```

Book Service

```
public class BookService
{
    private readonly IBookRepository bookRepository;

    public BookService(IBookRepository bookRepository)
    {
        this.bookRepository = bookRepository;
    }

    public BookModel GetById(int id)
    {
        var book = bookRepository.GetById(id);

        return Map(book);
    }

    public IReadOnlyCollection<BookModel> GetAllByQuery(string query)
    {
        var books = Book.IsIsbn(query)
            ? bookRepository.GetAllByIsbn(query)
            : bookRepository.GetAllByTitleOrAuthor(query);

        return books.Select(Map)
            .ToArray();
    }

    private BookModel Map(Book book)
    {
        return new BookModel
        {
            Id = book.Id,
            Isbn = book.Isbn,
            Title = book.Title,
            Author = book.Author,
            Description = book.Description,
            Price = book.Price,
        }
    }
}
```

```
};
}
}
```

Cart

```
public class Cart
{
    public int OrderId { get; }

    public int TotalCount { get; }

    public decimal TotalPrice { get; }

    public Cart(int orderId, int totalCount, decimal totalPrice)
    {
        OrderId = orderId;
        TotalCount = totalCount;
        TotalPrice = totalPrice;
    }
}
```

Order Item Model

```
public class OrderItemModel // Data Transfer Object DTO
{
    public int BookId { get; set; }

    public string Title { get; set; }

    public string Author { get; set; }

    public int Count { get; set; }

    public decimal Price { get; set; }
}
```

Order Model

```
public class OrderModel
{
    public int Id { get; set; }

    public OrderItemModel[] Items { get; set; } = new OrderItemModel[0];

    public int TotalCount { get; set; }

    public decimal TotalPrice { get; set; }

    public string CellPhone { get; set; }

    public string DeliveryDescription { get; set; }

    public string PaymentDescription { get; set; }

    public Dictionary<string, string> Errors { get; set; } = new Dictionary<string, string>();
}
```

```

public class OrderService
{
    private readonly IBookRepository bookRepository;
    private readonly IOrderRepository orderRepository;
    private readonly INotificationService notificationService;
    private readonly IHttpContextAccessor httpContextAccessor;

    protected ISession Session => httpContextAccessor.HttpContext.Session;

    public OrderService(IBookRepository bookRepository,
        IOrderRepository orderRepository,
        INotificationService notificationService,
        IHttpContextAccessor httpContextAccessor)
    {
        this.bookRepository = bookRepository;
        this.orderRepository = orderRepository;
        this.notificationService = notificationService;
        this.httpContextAccessor = httpContextAccessor;
    }

    public bool TryGetModel(out OrderModel model)
    {
        if (TryGetOrder(out Order order))
        {
            model = Map(order);
            return true;
        }

        model = null;
        return false;
    }

    internal bool TryGetOrder(out Order order)
    {
        if (Session.TryGetCart(out Cart cart))
        {
            order = orderRepository.GetById(cart.OrderId);
            return true;
        }

        order = null;
        return false;
    }

    internal OrderModel Map(Order order)
    {
        var books = GetBooks(order);
        var items = from item in order.Items
            join book in books on item.BookId equals book.Id
            select new OrderItemModel
            {
                BookId = book.Id,
                Title = book.Title,
                Author = book.Author,
                Price = item.Price,
                Count = item.Count,
            };
    }
}

```

```

return new OrderModel
{
    Id = order.Id,
    Items = items.ToArray(),
    TotalCount = order.TotalCount,
    TotalPrice = order.TotalPrice,
    CellPhone = order.CellPhone,
    DeliveryDescription = order.Delivery?.Description,
    PaymentDescription = order.Payment?.Description // .? оператор проверки на null
};
}

internal IEnumerable<Book> GetBooks(Order order)
{
    var bookIds = order.Items.Select(item => item.BookId);

    return bookRepository.GetAllByIds(bookIds);
}

public OrderModel AddBook(int bookId, int count)
{
    if (count < 1)
        throw new InvalidOperationException("Too few books to add");

    if (!TryGetOrder(out Order order))
        order = orderRepository.Create();

    AddOrUpdateBook(order, bookId, count);
    UpdateSession(order);

    return Map(order);
}

internal void AddOrUpdateBook(Order order, int bookId, int count)
{
    var book = bookRepository.GetById(bookId);

    if (order.Items.TryGet(bookId, out OrderItem orderItem))
        orderItem.Count += count;
    else
        order.Items.Add(book.Id, book.Price, count);

    orderRepository.Update(order);
}

internal void UpdateSession(Order order)
{
    var cart = new Cart(order.Id, order.TotalCount, order.TotalPrice);
    Session.Set(cart);
}

public OrderModel UpdateBook(int bookId, int count)
{
    var order = GetOrder();
    order.Items.Get(bookId).Count = count;

    orderRepository.Update(order);
    UpdateSession(order);

    return Map(order);
}

```

```

}

public OrderModel RemoveBook(int bookId)
{
    var order = GetOrder();
    order.Items.Remove(bookId);

    orderRepository.Update(order);
    UpdateSession(order);

    return Map(order);
}

public Order GetOrder()
{
    if (TryGetOrder(out Order order))
        return order;

    throw new InvalidOperationException("Empty session.");
}

public OrderModel SendConfirmation(string cellPhone)
{
    var order = GetOrder();
    var model = Map(order);

    if (TryFormatPhone(cellPhone, out string formattedPhone))
    {
        var confirmationCode = 1111; // todo: random.Next(1000, 10000) = 1000, 1001, ..., 9998, 9999
        model.CellPhone = formattedPhone;
        Session.SetInt32(formattedPhone, confirmationCode);
        notificationService.SendConfirmationCode(formattedPhone, confirmationCode);
    }
    else
        model.Errors["cellPhone"] = "Номер телефона не соответствует формату +79876543210";

    return model;
}

private readonly PhoneNumberUtil phoneNumberUtil = PhoneNumberUtil.GetInstance();

internal bool TryFormatPhone(string cellPhone, out string formattedPhone)
{
    try // Библиотека с номерами телефонов не поддерживает метод TryParse, поэтому мы вынуждены
        использовать try catch
    {
        var phoneNumber = phoneNumberUtil.Parse(cellPhone, "ru");
        formattedPhone = phoneNumberUtil.Format(phoneNumber, PhoneNumberFormat.INTERNATIONAL);
        return true;
    }
    catch (NumberParseException)
    {
        formattedPhone = null;
        return false;
    }
}

public OrderModel ConfirmCellPhone(string cellPhone, int confirmationCode)
{
    int? storedCode = Session.GetInt32(cellPhone);

```



```

var model = new OrderModel();

if (storedCode == null)
{
    model.Errors["cellPhone"] = "Что-то случилось. Попробуйте получить код ещё раз.";
    return model;
}

if (storedCode != confirmationCode)
{
    model.Errors["confirmationCode"] = "Неверный код. Проверьте и попробуйте ещё раз.";
    return model;
}

var order = GetOrder();
order.CellPhone = cellPhone;
orderRepository.Update(order);

Session.Remove(cellPhone);

return Map(order);
}

public OrderModel SetDelivery(OrderDelivery delivery)
{
    var order = GetOrder();
    order.Delivery = delivery;
    orderRepository.Update(order);

    return Map(order);
}

public OrderModel SetPayment(OrderPayment payment)
{
    var order = GetOrder();
    order.Payment = payment;
    orderRepository.Update(order);
    Session.RemoveCart();

    notificationService.StartProcess(order);

    return Map(order);
}
}

```

Session Extensions

```

public static class SessionExtensions
{
    private const string key = "Cart";

    public static void RemoveCart(this ISession session)
    {
        session.Remove(key);
    }

    public static void Set(this ISession session, Cart value)
    {
        if (value == null) // Если в корзине ничего нет - мы её не сохраняем
            return;

        using (var stream = new MemoryStream())

```

```

        using (var writer = new BinaryWriter(stream, Encoding.UTF8, true)) // Адаптер. Паттерн приведения одного
        интерфейса к другому
        {
            writer.Write(value.OrderId);
            writer.Write(value.TotalCount);
            writer.Write(value.TotalPrice);

            session.Set(key, stream.ToArray());
        }
    }

    public static bool TryGetCart(this ISession session, out Cart value)
    {
        if(session.TryGetValue(key, out byte[] buffer))
        {
            using (var stream = new MemoryStream(buffer))
            using (var reader = new BinaryReader(stream, Encoding.UTF8, true))
            {
                var orderId = reader.ReadInt32();
                var totalCount = reader.ReadInt32();
                var totalPrice = reader.ReadDecimal();

                value = new Cart(orderId, totalCount, totalPrice);

                return true;
            }
        }

        value = null;
        return false;
    }
}

```

Book

```

public class Book
{
    private readonly BookDto dto;

    //public int Id => dto.Id
    public int Id { get { return dto.Id; } }

    public string Isbn
    {
        get => dto.Isbn;
        set
        {
            if (TryFormatIsbn(value, out string formattedIsbn))
                dto.Isbn = formattedIsbn;

            throw new ArgumentException(nameof(Isbn));
        }
    }

    public string Author
    {
        get => dto.Author;
        set => dto.Author = value?.Trim();
    }

    public string Title

```

```

{
    get => dto.Title;
    set
    {
        if (string.IsNullOrEmpty(value))
            throw new ArgumentException(nameof(Title));

        dto.Title = value.Trim();
    }
}

public string Description
{
    get => dto.Description;
    set => dto.Description = value;
}

public decimal Price
{
    get => dto.Price;
    set => dto.Price = value;
}

internal Book(BookDto dto)
{
    this.dto = dto;
}

public static bool TryFormatIsbn(string isbn, out string formattedIsbn)
{
    if (isbn == null)
    {
        formattedIsbn = null;
        return false;
    }

    formattedIsbn = isbn.Replace("-", "")
        .Replace(" ", "")
        .ToUpper();

    return Regex.IsMatch(formattedIsbn, @"^ISBN\d{10}(\d{3})?$");
}

public static bool IsIsbn(string isbn)
    => TryFormatIsbn(isbn, out _);

public static class DtoFactory
{
    public static BookDto Create(string isbn,
        string author,
        string title,
        string description,
        decimal price)
    {
        if (TryFormatIsbn(isbn, out string formattedIsbn))
            isbn = formattedIsbn;
        else
            throw new ArgumentException(nameof(isbn));

        if (string.IsNullOrEmpty(title))

```

```

        throw new ArgumentException(nameof(title));

        return new BookDto
        {
            Isbn = isbn,
            Author = author?.Trim(),
            Title = title.Trim(),
            Description = description?.Trim(),
            Price = price,
        };
    }
}

public static class Mapper
{
    public static Book Map(BookDto dto) => new Book(dto);

    public static BookDto Map(Book domain) => domain.dto;
}

```

Order

```

public class Order // Сущность
{
    private readonly OrderDto dto;

    public int Id => dto.Id;

    public string CellPhone
    {
        get => dto.CellPhone;
        set
        {
            if (string.IsNullOrEmpty(value))
                throw new ArgumentException(nameof(CellPhone));

            dto.CellPhone = value;
        }
    }

    public OrderDelivery Delivery
    {
        get
        {
            if (dto.DeliveryUniqueCode == null)
                return null;

            return new OrderDelivery(
                dto.DeliveryUniqueCode,
                dto.DeliveryDescription,
                dto.DeliveryPrice,
                dto.DeliveryParameters);
        }
        set
        {
            if (value == null)
                throw new ArgumentException(nameof(Delivery));

            dto.DeliveryUniqueCode = value.UniqueCode;
            dto.DeliveryDescription = value.Description;
        }
    }
}

```

```

        dto.DeliveryPrice = value.Price;
        dto.DeliveryParameters = value.Parameters
            .ToDictionary(p => p.Key, p => p.Value);
    }
}

public OrderPayment Payment
{
    get
    {
        if (dto.PaymentServiceName == null)
            return null;

        return new OrderPayment(
            dto.PaymentServiceName,
            dto.PaymentDescription,
            dto.PaymentParameters);
    }
    set
    {
        if (value == null)
            throw new ArgumentException(nameof(Payment));

        dto.PaymentServiceName = value.UniqueCode;
        dto.PaymentDescription = value.Description;
        dto.PaymentParameters = value.Parameters
            .ToDictionary(p => p.Key, p => p.Value);
    }
}

public OrderItemCollection Items { get; }

public int TotalCount => Items.Sum(item => item.Count);

public decimal TotalPrice => Items.Sum(item => item.Price * item.Count)
    + (Delivery?.Price ?? 0m);

public Order(OrderDto dto)
{
    this.dto = dto;
    Items = new OrderItemCollection(dto);
}

public static class DtoFactory
{
    public static OrderDto Create() => new OrderDto();
}

public static class Mapper
{
    public static Order Map(OrderDto dto) => new Order(dto);

    public static OrderDto Map(Order domain) => domain.dto;
}
}

```

Order Delivery

```
public class OrderDelivery
{
    public string UniqueCode { get; }

    public string Description { get; }

    public decimal Price { get; } // Стоимость платной доставки

    public IReadOnlyDictionary<string, string> Parameters { get; }

    public OrderDelivery(string uniqueCode, string description, decimal amount, IReadOnlyDictionary<string, string>
parameters)
    {
        if (string.IsNullOrEmpty(uniqueCode))
            throw new ArgumentException(nameof(uniqueCode));

        if (string.IsNullOrEmpty(description))
            throw new ArgumentException(nameof(description));

        if (parameters == null)
            throw new ArgumentNullException(nameof(parameters));

        UniqueCode = uniqueCode;
        Description = description;
        Price = amount;
        Parameters = parameters;
    }
}
```

Order Item

```
public class OrderItem
{
    private readonly OrderItemDto dto;

    public int BookId => dto.BookId;

    public int Count
    {
        get { return dto.Count; }
        set
        {
            ThrowIfInvalidCount(value);

            dto.Count = value;
        }
    }

    public decimal Price
    {
        get => dto.Price;
        set => dto.Price = value;
    }

    internal OrderItem(OrderItemDto dto)
    {
        this.dto = dto;
    }

    private static void ThrowIfInvalidCount(int count)
```

```

{
    if (count <= 0)
        throw new ArgumentOutOfRangeException("Count must be greater than zero");
}

public static class DtoFactory
{
    public static OrderItemDto Create(OrderDto order, int bookId, decimal price, int count)
    {
        if (order == null)
            throw new ArgumentNullException(nameof(order));

        ThrowIfInvalidCount(count);

        return new OrderItemDto
        {
            BookId = bookId,
            Price = price,
            Count = count,
            Order = order,
        };
    }
}

public static class Mapper
{
    public static OrderItem Map(OrderItemDto dto) => new OrderItem(dto);

    public static OrderItemDto Map(OrderItem domain) => domain.dto;
}
}

```

Order Item Collection

```

public class OrderItemCollection : IReadOnlyCollection<OrderItem>
{
    private readonly OrderDto orderDto;
    private readonly List<OrderItem> items;

    public OrderItemCollection(OrderDto orderDto)
    {
        if (orderDto == null)
            throw new ArgumentNullException(nameof(orderDto));

        this.orderDto = orderDto;

        items = orderDto.Items
            .Select(OrderItem.Mapper.Map)
            .ToList();
    }

    public int Count => items.Count;

    public IEnumerator<OrderItem> GetEnumerator()
    {
        return items.GetEnumerator();
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return (items as IEnumerable).GetEnumerator();
    }
}

```



```

}

public OrderItem Get(int bookId)
{
    if (TryGet(bookId, out OrderItem orderItem))
        return orderItem;

    throw new InvalidOperationException("Book not found.");
}

public bool TryGet(int bookId, out OrderItem orderItem)
{
    var index = items.FindIndex(item => item.BookId == bookId);
    if (index == -1)
    {
        orderItem = null;
        return false;
    }

    orderItem = items[index];
    return true;
}

public OrderItem Add(int bookId, decimal price, int count)
{
    if (TryGet(bookId, out OrderItem orderItem))
        throw new InvalidOperationException("Book already exists.");

    var orderItemDto = OrderItem.DtoFactory.Create(orderDto, bookId, price, count);
    orderDto.Items.Add(orderItemDto);

    orderItem = OrderItem.Mapper.Map(orderItemDto);
    items.Add(orderItem);

    return orderItem;
}

public void Remove(int bookId)
{
    var index = items.FindIndex(item => item.BookId == bookId);
    if (index == -1)
        throw new InvalidOperationException("Can't find book to remove from order.");

    orderDto.Items.RemoveAt(index);
    items.RemoveAt(index);
}
}

```

Order Payment

```

public class OrderPayment
{
    public string UniqueCode { get; }

    public string Description { get; }

    public IReadOnlyDictionary<string, string> Parameters { get; }

    public OrderPayment(string uniqueCode, string description, IReadOnlyDictionary<string, string> parameters)
    {
        if (string.IsNullOrEmpty(uniqueCode))
            throw new ArgumentException(nameof(uniqueCode));
    }
}

```

```

        if (string.IsNullOrEmpty(description))
            throw new ArgumentException(nameof(description));

        if (parameters == null)
            throw new ArgumentNullException(nameof(parameters));

        UniqueCode = uniqueCode;
        Description = description;
        Parameters = parameters;
    }
}

```

Book Tests

```

public class BookTests
{
    [Fact]
    public void IsIsbn_WithNull_ReturnFalse()
    {
        bool actual = Book.IsIsbn(null); // Если ф-ия IsIsbn запускается с параметром null, то она должна вернуть
false

        Assert.False(actual);
    }

    [Fact]
    public void IsIsbn_WithBlankString_ReturnFalse() // С пустой строкой
    {
        bool actual = Book.IsIsbn(" ");

        Assert.False(actual);
    }

    [Fact]
    public void IsIsbn_WithInvalidIsbn_ReturnFalse() // С неверным isbn
    {
        bool actual = Book.IsIsbn("ISBN 123");

        Assert.False(actual);
    }

    [Fact]
    public void IsIsbn_WithIsbn10_ReturnTrue() // С 10 цифрами в Isbn, с неограниченным кол-вом дефисов и
регистром
    {
        bool actual = Book.IsIsbn("IsBn 123-456-789 0");

        Assert.True(actual);
    }

    [Fact]
    public void IsIsbn_WithIsbn13_ReturnTrue() // Если isbn содержит больше цифр (13) - всё ок
    {
        bool actual = Book.IsIsbn("IsBn 123-456-789 0123");

        Assert.True(actual);
    }

    [Fact]
    public void IsIsbn_WithTrashStart_ReturnFalse()
    {

```

```
bool actual = Book.IsIsbn("xxx IsBn 123-456-789 0123 yyy");
```

```
Assert.False(actual);
```

```
}
```

```
}
```

Order Item Collection Tests

```
public class OrderItemCollectionTests
```

```
{
```

```
[Fact]
```

```
public void Get_WithExistingItem_ReturnsItem()
```

```
{
```

```
var order = CreateTestOrder();
```

```
var orderItem = order.Items.Get(1);
```

```
Assert.Equal(3, orderItem.Count);
```

```
}
```

```
private static Order CreateTestOrder()
```

```
{
```

```
return new Order(new OrderDto
```

```
{
```

```
Id = 1,
```

```
Items = new List<OrderItemDto>
```

```
{
```

```
new OrderItemDto { BookId = 1, Price = 10m, Count = 3},
```

```
new OrderItemDto { BookId = 2, Price = 100m, Count = 5},
```

```
}
```

```
});
```

```
}
```

```
[Fact]
```

```
public void Get_WithNonExistingItem_ThrowsInvalidOperationException()
```

```
{
```

```
var order = CreateTestOrder();
```

```
Assert.Throws<InvalidOperationException>(() =>
```

```
{
```

```
order.Items.Get(100);
```

```
});
```

```
}
```

```
[Fact]
```

```
public void Add_WithExistingItem_ThrowInvalidOperationException()
```

```
{
```

```
var order = CreateTestOrder();
```

```
Assert.Throws<InvalidOperationException>(() =>
```

```
{
```

```
order.Items.Add(1, 10m, 10);
```

```
});
```

```
}
```

```
[Fact]
```

```
public void Add_WithNewItem_SetsCount()
```

```
{
```

```
var order = CreateTestOrder();
```

```
order.Items.Add(4, 30m, 10);
```

```

    Assert.Equal(10, order.Items.Get(4).Count);
}

[Fact]
public void Remove_WithExistingItem_RemovesItem()
{
    var order = CreateTestOrder();

    order.Items.Remove(1);

    Assert.Collection(order.Items,
        item => Assert.Equal(2, item.BookId));
}

```

```

[Fact]
public void Remove_WithNonExistingItem_ThrowsInvalidOperationException()
{
    var order = CreateTestOrder();

    Assert.Throws<InvalidOperationException>(() =>
    {
        order.Items.Remove(100);
    });
}
}

```

Order Item Tests

```

public class OrderItemTests
{
    [Fact]
    public void OrderItem_WithZeroCount_ThrowArgumentOutOfRangeException()
    {
        Assert.Throws<ArgumentOutOfRangeException>(() =>
        {
            int count = 0;
            OrderItem.DtoFactory.Create(new OrderDto(), 1, 10m, count);
        });
    }

    [Fact]
    public void OrderItem_WithNegativeCount_ThrowArgumentOutOfRangeException()
    {
        Assert.Throws<ArgumentOutOfRangeException>(() =>
        {
            int count = -1;
            OrderItem.DtoFactory.Create(new OrderDto(), 1, 10m, count);
        });
    }

    [Fact]
    public void OrderItem_WithPositiveCount_SetsCount()
    {
        var orderItem = OrderItem.DtoFactory.Create(new OrderDto(), 1, 10m, 2);

        Assert.Equal(1, orderItem.BookId);
        Assert.Equal(10m, orderItem.Price);
        Assert.Equal(2, orderItem.Count);
    }

    [Fact]

```

```

public void Count_WithNegativeValue_ThrowsArgumentOutOfRangeException()
{
    var orderItemDto = OrderItem.DtoFactory.Create(new OrderDto(), 1, 10m, 30);
    var orderItem = OrderItem.Mapper.Map(orderItemDto);

    Assert.Throws<ArgumentOutOfRangeException>(() =>
    {
        orderItem.Count = -1;
    });
}

```

```

[Fact]
public void Count_WithZeroValue_ThrowsArgumentOutOfRangeException()
{
    var orderItemDto = OrderItem.DtoFactory.Create(new OrderDto(), 1, 10m, 30);
    var orderItem = OrderItem.Mapper.Map(orderItemDto);

    Assert.Throws<ArgumentOutOfRangeException>(() =>
    {
        orderItem.Count = 0;
    });
}

```

```

[Fact]
public void Count_WithPositiveValue_SetsValues()
{
    var orderItemDto = OrderItem.DtoFactory.Create(new OrderDto(), 1, 10m, 30);
    var orderItem = OrderItem.Mapper.Map(orderItemDto);

    orderItem.Count = 10;

    Assert.Equal(10, orderItem.Count);
}
}

```

Order Tests

```

public class OrderTests
{
    [Fact]
    public void TotalCount_WithEmptyItems_ReturnZero()
    {
        var order = CreateEmptyTestOrder();

        Assert.Equal(0, order.TotalCount);
    }

    private static Order CreateEmptyTestOrder()
    {
        return new Order(new OrderDto
        {
            Id = 1,
            Items = new OrderItemDto[0]
        });
    }

    [Fact]
    public void TotalPrice_WithEmptyItems_ReturnZero()
    {
        var order = CreateEmptyTestOrder();
    }
}

```

```

    Assert.Equal(0m, order.TotalCount);
}

```

```

private static Order CreateTestOrder()
{
    return new Order(new OrderDto
    {
        Id = 1,
        Items = new[]
        {
            new OrderItemDto { BookId = 1, Price = 10m, Count = 3},
            new OrderItemDto { BookId = 2, Price = 100m, Count = 5},
        }
    });
}

```

```

[Fact]
public void TotalCount_WithNonEmptyItems_CalculatesTotalCount()
{
    var order = CreateTestOrder();

    Assert.Equal(3 + 5, order.TotalCount);
}

```

```

[Fact]
public void TotalPrice_WithNonEmptyItems_CalculatesTotalPrice()
{
    var order = CreateTestOrder();

    Assert.Equal(3 * 10m + 5 * 100m, order.TotalPrice);
}
}

```

Stub Book Repository

```

public class OrderTests
class StubBookRepository : IBookRepository
{
    public Book[] ResultOfGetAllByIsbn { get; set; }

    public Book[] ResultOfGetAllByTitleOrAuthor { get; set; }

    public Book[] GetAllByIds(IEnumerable<int> bookIds)
    {
        throw new NotImplementedException();
    }

    public Book[] GetAllByIsbn(string isbn)
    {
        return ResultOfGetAllByIsbn;
    }

    public Book[] GetAllByTitleOrAuthor(string titleOrAuthor)
    {
        return ResultOfGetAllByTitleOrAuthor;
    }

    public Book GetById(int id)
    {
        throw new NotImplementedException();
    }
}

```

```

class BookRepository : IBookRepository
{
    private readonly DbContextFactory dbContextFactory;

    public BookRepository(DbContextFactory dbContextFactory)
    {
        this.dbContextFactory = dbContextFactory;
    }

    public Book[] GetAllByIds(IEnumerable<int> bookIds)
    {
        var dbContext = dbContextFactory.Create(typeof(BookRepository));

        return dbContext.Books
            .Where(book => bookIds.Contains(book.Id))
            .AsEnumerable()
            .Select(Book.Mapper.Map)
            .ToArray();
    }

    public Book[] GetAllByIsbn(string isbn)
    {
        var dbContext = dbContextFactory.Create(typeof(BookRepository));

        if (Book.TryFormatIsbn(isbn, out string formattedIsbn))
        {
            return dbContext.Books
                .Where(book => book.Isbn == formattedIsbn)
                .AsEnumerable()
                .Select(Book.Mapper.Map)
                .ToArray();
        }

        return new Book[0];
    }

    public Book[] GetAllByTitleOrAuthor(string titleOrAuthor)
    {
        var dbContext = dbContextFactory.Create(typeof(BookRepository));

        var parameter = new SqlParameter("@titleOrAuthor", titleOrAuthor);

        return dbContext.Books
            .Where(book => book.Author.Contains(titleOrAuthor)
                || book.Title.Contains(titleOrAuthor))
            .AsEnumerable()
            .Select(Book.Mapper.Map)
            .ToArray();

        // Реализация с полномасштабным поиском не работает :(
        //return dbContext.Books
        //    .FromSqlRaw("SELECT * FROM Books WHERE CONTAINS((Author, Title), @titleOrAuthor)",
        //        parameter)
        //    .AsEnumerable()
        //    .Select(Book.Mapper.Map)
        //    .ToArray();
    }

    public Book GetById(int id)

```

```

{
    var dbContext = dbContextFactory.Create(typeof(BookRepository));

    var dto = dbContext.Books
        .Single(book => book.Id == id);

    return Book.Mapper.Map(dto);
}

```

Db Context Factory

```

class DbContextFactory
{
    private readonly IHttpContextAccessor httpContextAccessor;

    public DbContextFactory(IHttpContextAccessor httpContextAccessor)
    {
        this.httpContextAccessor = httpContextAccessor;
    }

    public StoreDbContext Create(Type repositoryType)
    {
        var services = httpContextAccessor.HttpContext.RequestServices;

        var dbContexts = services.GetService<Dictionary<Type, StoreDbContext>>();
        if (!dbContexts.ContainsKey(repositoryType))
            dbContexts[repositoryType] = services.GetService<StoreDbContext>();

        return dbContexts[repositoryType];
    }
}

```

Order Repository

```

class OrderRepository : IOrderRepository
{
    private readonly DbContextFactory dbContextFactory;

    public OrderRepository(DbContextFactory dbContextFactory)
    {
        this.dbContextFactory = dbContextFactory;
    }

    public Order Create()
    {
        var dbContext = dbContextFactory.Create(typeof(OrderRepository));

        var dto = Order.DtoFactory.Create();
        dbContext.Orders.Add(dto);
        dbContext.SaveChanges();

        return Order.Mapper.Map(dto);
    }

    public Order GetById(int id)
    {
        var dbContext = dbContextFactory.Create(typeof(OrderRepository));

        var dto = dbContext.Orders
            .Include(order => order.Items)
            .Single(order => order.Id == id);

        return Order.Mapper.Map(dto);
    }
}

```



```

    }

    public void Update(Order order)
    {
        var dbContext = dbContextFactory.Create(typeof(OrderRepository));

        dbContext.SaveChanges();
    }
}

```

Service Collection Extensions

```

public static class ServiceCollectionExtensions
{
    public static IServiceCollection AddEfRepositories(this IServiceCollection services, string connectionString)
    {
        services.AddDbContext<StoreDbContext>(
            options =>
            {
                options.UseSqlServer(connectionString);
            },
            ServiceLifetime.Transient
        );

        services.AddScoped<Dictionary<Type, StoreDbContext>>();
        services.AddSingleton<DbContextFactory>();

        services.AddSingleton<IBookRepository, BookRepository>();
        services.AddSingleton<IOrderRepository, OrderRepository>();

        return services;
    }
}

```

Store Db Context

```

public class StoreDbContext : DbContext
{
    public DbSet<BookDto> Books { get; set; }

    public DbSet<OrderDto> Orders { get; set; }

    public DbSet<OrderItemDto> OrderItems { get; set; }

    public StoreDbContext(DbContextOptions<StoreDbContext> options)
        : base(options)
    { }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // Будем для BookDto описывать параметры свойств, то есть будет описана таблица, в таблице будут
        книги
        BuildBooks(modelBuilder);

        BuildOrders(modelBuilder);

        BuildOrderItems(modelBuilder);
    }

    private void BuildOrderItems(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<OrderItemDto>(action =>
        {
            action.Property(dto => dto.Price)

```

```

        .HasColumnType("money");

        action.HasOne(dto => dto.Order) // К одному Order может относиться несколько OrderItems
            .WithMany(dto => dto.Items)
            .IsRequired();
    });
}

private static void BuildOrders(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<OrderDto>(action =>
    {
        action.Property(dto => dto.CellPhone)
            .HasMaxLength(20);

        action.Property(dto => dto.DeliveryUniqueCode)
            .HasMaxLength(40);

        action.Property(dto => dto.DeliveryPrice)
            .HasColumnType("money");

        action.Property(dto => dto.PaymentServiceName)
            .HasMaxLength(40);

        // Добавляем свойство и говорим, что у него есть конверсия, он берёт на вход словарь, что-то с ним
        // сделает, и вернёт строку
        // но ему для работы нужен ValueComparer. Для того, чтобы Entity Framework должен проверять словари
        // на совпадение, мы
        // предоставляем ему этот ValueComparer
        action.Property(dto => dto.DeliveryParameters)
            .HasConversion(
                value => JsonConvert.SerializeObject(value),
                value => JsonConvert.DeserializeObject<Dictionary<string, string>>(value))
            .Metadata.SetValueComparer(DictionaryComparer);

        action.Property(dto => dto.PaymentParameters)
            .HasConversion(
                value => JsonConvert.SerializeObject(value),
                value => JsonConvert.DeserializeObject<Dictionary<string, string>>(value))
            .Metadata.SetValueComparer(DictionaryComparer);
    });
}

// Тут 2 метода. Первый который сравнивает две последовательности, а второй умеет считать GetHashCode
private static readonly ValueComparer DictionaryComparer =
    new ValueComparer<Dictionary<string, string>>(
        (dictionary1, dictionary2) => dictionary1.SequenceEqual(dictionary2),
        dictionary => dictionary.Aggregate(
            0,
            (a, p) => GetHashCode.Combine(HashCodes.Combine(a, p.Key.GetHashCode()), p.Value.GetHashCode())
        )
    );

private static void BuildBooks(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<BookDto>(action =>
    {
        action.Property(dto => dto.Isbn)
            .HasMaxLength(17)
            .IsRequired();
    });
}

```

```

        action.Property(dto => dto.Title)
            .IsRequired();

        action.Property(dto => dto.Price)
            .HasColumnType("money");

        action.HasData(
            new BookDto
            {
                Id = 1,
                Isbn = "ISBN0201038013",
                Author = "D. Knuth",
                Title = "Art Of Programming, Vol. 1",
                Description = "This volume begins with basic programming concepts and techniques, then focuses more
particularly on information structures-the representation of information inside a computer, the structural relationships
between data elements and how to deal with them efficiently.",
                Price = 7.19m,
            },
            new BookDto
            {
                Id = 2,
                Isbn = "ISBN0201485672",
                Author = "M. Fowler",
                Title = "Refactoring",
                Description = "As the application of object technology--particularly the Java programming language--has
become commonplace, a new problem has emerged to confront the software development community.",
                Price = 12.45m,
            },
            new BookDto
            {
                Id = 3,
                Isbn = "ISBN0131101633",
                Author = "B. W. Kernighan, D. M. Ritchie",
                Title = "C Programming Language",
                Description = "Known as the bible of C, this classic bestseller introduces the C programming language and
illustrates algorithms, data structures, and programming techniques.",
                Price = 14.98m,
            },
            new BookDto
            {
                Id = 4,
                Isbn = "ISBN9780132350884",
                Author = "Robert C. Martin",
                Title = "Clean Code: A Handbook of Agile Software Craftsmanship 1st Edition",
                Description = "Even bad code can function. But if code isn't clean, it can bring a development organization to
its knees. Every year, countless hours and significant resources are lost because of poorly written code. But it doesn't have
to be that way.",
                Price = 26.29m,
            }
        );
    });
}
}

```

Yandex Kassa Payment Service

```

public class YandexKassaPaymentService : IPaymentService, IWebContractorService
{
    private readonly IHttpContextAccessor httpContextAccessor;

    public YandexKassaPaymentService(IHttpContextAccessor httpContextAccessor)
    {
    }
}

```

```

{
    this.httpContextAccessor = httpContextAccessor;
}

private HttpRequest Request => httpContextAccessor.HttpContext.Request;

public string Name => "YandexKassa";

public string Title => "Оплата банковской картой";

public Form FirstForm(Order order)
{
    return Form.CreateFirst(Name)
        .AddParameter("orderId", order.Id.ToString());
}

public Form NextForm(int step, IReadOnlyDictionary<string, string> values)
{
    if (step != 1)
        throw new InvalidOperationException("Invalid Yandex.Kassa payment step.");

    return Form.CreateLast(Name, step + 1, values);
}

public OrderPayment GetPayment(Form form)
{
    if (form.ServiceName != Name || !form.IsFinal)
        throw new InvalidOperationException("Invalid payment form.");

    return new OrderPayment(Name, "Оплатой картой", form.Parameters);
}

public Uri StartSession(IReadOnlyDictionary<string, string> parameters, Uri returnUrl)
{
    var queryString = QueryString.Create(parameters);
    queryString += QueryString.Create("returnUri", returnUrl.ToString());

    var builder = new UriBuilder(Request.Scheme, Request.Host.Host)
    {
        Path = "YandexKassa/",
        Query = queryString.ToString(),
    };

    if (Request.Host.Port != null)
        builder.Port = Request.Host.Port.Value;

    return builder.Uri;
}
}

```

Extension Filters

```

public class ExceptionFilter : IExceptionFilter
{
    private readonly IWebHostEnvironment webHostEnvironment;

    public ExceptionFilter(IWebHostEnvironment webHostEnvironment)
    {
        this.webHostEnvironment = webHostEnvironment;
    }

    public void OnException(ExceptionContext context)

```

```

{
    if (webHostEnvironment.IsDevelopment())
        return;

    if (context.Exception.TargetSite.Name == "ThrowNoElementsException")
    {
        context.Result = new ViewResult
        {
            ViewName = "NotFound",
            StatusCode = 404,
        };
    }
}
}

```

Program

```

public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}

```

Startup

```

public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews(options =>
        {
            options.Filters.Add(typeof(ExceptionFilter));
        });

        services.AddHttpContextAccessor();

        services.AddDistributedMemoryCache(); // Наша корзина будет храниться в дистрибутивной памяти
        services.AddSession(options =>
        {
            options.IdleTimeout = TimeSpan.FromMinutes(20); // Время жизни сессии
            options.Cookie.HttpOnly = true;
            options.Cookie.IsEssential = true; // Хранение информации о пользователях по соглашению JD
        });

        services.AddEfRepositories(Configuration.GetConnectionString("Store"));

        services.AddSingleton<INotificationService, DebugNotificationService>();
    }
}

```

```

services.AddSingleton<IDeliveryService, PostamateDeliveryService>();
services.AddSingleton<IPaymentService, CashPaymentService>();
services.AddSingleton<IPaymentService, YandexKassaPaymentService>();
services.AddSingleton<IWebContractorService, YandexKassaPaymentService>();
services.AddSingleton<BookService>();
services.AddSingleton<OrderService>();
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    // if (env.IsDevelopment())
    if (false)
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days. You may want to change this for production scenarios, see
        https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseSession();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "areas",
            pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}");

        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}"); // По умолчанию если задан controller, то он всегда
            // Home, а если не задано действие, то это index
    });
}
}

```

Уменьшенный вариант схемы на A1

