Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: компьютерного проектирования   
Кафедра: экономической информатики

Дисциплина: Средства и технологии анализа и разработки информационных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту  
на тему

«**РАЗРАБОТКА ИНТЕРНЕТ-МАГАЗИНА БЫТОВОЙ ТЕХНИКИ**»

Выполнил:

Студент группы 814302 Марчук А.А.

Проверил:

Ассистент кафедры ЭИ Г.П. Котковец

Минск 2022

**РЕФЕРАТ**

БГУИР КР 1-40 05 01 011 ПЗ

**Лизунова, В.О.** Разработка интернет-магазина бытовой техники: пояснительная записка к курсовой работе / В.О. Лизунова. – Минск: БГУИР, 2021. – 77 с.

Пояснительная записка 75 с., 41 рис., 11 источников,  
3 приложения

ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА. БАЗА ДАННЫХ. ИНТЕРНЕТ-МАГАЗИН БЫТОВОЙ ТЕХНИКИ. ОПТИМИЗАЦИЯ. МОДЕЛИРОВАНИЕ ДИАГРАММ. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС.

*Цель* *проектирования*: разработка программного средства для осуществления онлайн торговли. Программное средство должно предоставлять возможность выбирать товары, оформлять заказы и отслеживать из статус, позволяя пользователю управлять процессом покупки выбранных единиц продукции.

*Методология проведения работы*: системные, статистические и общенаучные методы.

*Результаты работы*: была разработана подробная функциональная модель программного средства. В процессе проектирования также были реализованы и описаны модели представления программного средства, что помогло его создать. Подробно проанализированы требования, необходимые для успешной реализации, разработана графическая часть. По итогу были реализованы серверная и клиентская части программного средства, позволяющие пользователю взаимодействовать с ним, создана база данных, используемая для хранения всей необходимой информации для успешного функционирования.

В процессе проектирования был разработан удобный пользовательский интерфейс, который позволяет выполнять действия, необходимые для реализации цели работы.

*Область применения результатов*: разработанное программное средство решает задачу по сокращению времени выбора товаров, оптимизации штата, увеличению объёма продаж, а также расширению клиентской базы.

**СОДЕРЖАНИЕ**

[Введение 6](#_heading=h.1fob9te)

[1 Анализ и моделирование предметной области программного средства 7](#_heading=h.3znysh7)

[1.1. Описание предметной области 7](#_heading=h.2et92p0)

[1.2. Разработка функциональной модели предметной области 9](#_heading=h.tyjcwt)

[1.3. Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований 18](#_heading=h.3dy6vkm)

[1.4. Разработка информационной модели предметной области 20](#_heading=h.4d34og8)

[1.5. Модели представления программного средства и их описание 27](#_heading=h.2s8eyo1)

[2 Проектирование и конструирование программного средства 31](#_heading=h.17dp8vu)

[2.1 Постановка задачи 31](#_heading=h.3rdcrjn)

[2.2 Архитектурные решения 32](#_heading=h.26in1rg)

[2.3 Описание алгоритмов, реализующих бизнес-логику разрабатываемого программного средства 34](#_heading=h.lnxbz9)

[2.4 Проектирование пользовательского интерфейса 38](#_heading=h.35nkun2)

[2.5 Обоснование выбора компонентов и технологий для реализации программного средства 42](#_heading=h.1ksv4uv)

[3 Тестирование и проверка работоспособности программного средства 48](#_heading=h.44sinio)

[4 Руководство по развертыванию и использованию программного средства 58](#_heading=h.2jxsxqh)

[Заключение 59](#_heading=h.z337ya)

[Список использованных источников 60](#_heading=h.3j2qqm3)

[Приложение А (обязательное) Отчет о проверке на заимствования в системе «Антиплагиат» 61](#_heading=h.1y810tw)

[Приложение Б (обязательное) Листинг кода алгоритмов, реализующих бизнес-логику 62](#_heading=h.4i7ojhp)

[Приложение В (обязательное) Листинг скрипта генерации базы данных 62](#_heading=h.2xcytpi)

**ВВЕДЕНИЕ**

В настоящее время Интернет становится все более развитой средой для осуществления коммуникаций с потребителями, а также инструментом для упрощения их жизней. Интернет становится удобным и достаточно доступным сервисом для поиска нужной информации.

В последние годы все большее развитие получает электронная коммерция. Этому способствует научно-технический прогресс в области информационных технологий и электроники. Электронная коммерция – термин, используемый для обозначения коммерческой активности в сети Интернет.

Одним из видов электронной коммерции является торговля посредством Интернет-магазина. Интернет-магазин – это сайт (портал), предлагающий товары и/или услуги для продажи и предоставляющий средства их заказа и оплаты.

За год число онлайн-заказов в интернет-магазинах электроники выросло на 37%. При этом за тот же период времени общая выручка, полученная магазинами с офлайн-продаж, увеличилась только лишь на 24%. Эти данные лучше всего отражают актуальность разработки онлайн-магазина для реализации продукции предприятия [1].

Использование онлайн-магазина для продажи электронных устройств позволит повысить прибыль предприятия, привлечь новых покупателей и значительно сократить расходы на обслуживание точек продажи товаров.

Объектом исследования являются веб-сервисы электронной торговли, а предметом – методы и инструментальные средства разработки программного обеспечения службы покупки товаров онлайн.

Целью курсового проекта является повышение эффективности работы магазина за счет упрощения процесса поиска и оформления заказа товаров, обеспечения удобной работы клиентов.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить предметную область курсового проекта.
2. Ознакомиться с современными технологиями разработки программных приложений и использовать их в своей работе.
3. Разработать информационную модель системы.
4. Продумать и реализовать алгоритмы бизнес-логики.
5. Спроектировать и создать базу данных.
6. Разработать серверную часть, взаимодействующую с базой данных.
7. Реализовать требуемый пользователю функционал.
8. **АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА** 
   1. **Описание предметной области**

Бытовая техника, оборудование и приборы, облегчающие ведение домашнего хозяйства благодаря его механизации. Эти приборы и устройства разрабатывались не в рамках какого-то определенного комплекта или в соответствии с каким-то планом, предусматривавшим некую последовательность, - они просто появлялись один за другим с последующим введением усовершенствований, которые повышали их надежность и эффективность по сравнению с уровнем, уже достигнутым ранее. В состав современного бытового электрооборудования входят холодильники и стиральные машины, пылесосы и электроутюги, радиоприемники и телевизоры, видеокамеры и диктофоны, микроволновые печи, медицинские приборы индивидуального пользования, калькуляторы и многие другие приборы и устройства, которые стали доступны в быту с развитием электроники и автоматики. Электродвигатели, нагреватели и охлаждающие устройства, встроенные в каркасы многих издавна используемых бытовых машин и механизмов, дали им новую жизнь; в целях безопасного использования они дополнительно снабжаются предохранительными клапанами и аварийными выключателями. Многие дополнительные детали и приспособления, такие, как таймеры, термореле, световые индикаторы печей и сенсорные панели, изготовители бытовой техники включают в комплект в целях автоматизации, удобства и улучшения внешнего вида.

Интернет, как средство массовой информации существует достаточно давно. Для большинства людей, использующих Интернет, это просто средство для получения какой-либо информации, либо средство коммуникации. Одним из лежащих на поверхности способов использования Интернета для бизнеса является Интернет-продажа товаров и услуг.

Развитие сети Интернет и появление электронной торговли вызывают существенные перемены в экономике и бизнесе, приводят к изменению традиционных положений экономической теории и практики. Технологии и стандарты передачи данных через сеть Интернет стали универсальной средой обмена коммерческой информацией, что во многом определило принципы ведения бизнеса в сфере электронной коммерции.

Республика Беларусь развивает электронную торговлю, как одно из важных направлений повышения конкурентоспособности предприятий и выхода на новые рынки товаров и услуг. Технологии и стандарты передачи данных через сеть Интернет стали универсальной средой обмена коммерческой информацией и во многом определили основные принципы ведения бизнеса в сфере электронной коммерции.

Поход по магазинам может отнимать большое количество времени и дополнительных затрат на дорогу, при этом нет гарантий, что приехав в магазин покупатель сразу найдет подходящий товар.

Автоматизированная система обслуживанию клиентов позволит сэкономить время на поиски нужного товара, а также оставить или прочитать отзывы на товары и самое главное не выходя из дома оформить доставку до подъезда.

Кроме того, хранение информации о клиентах позволяет в дальнейшем накапливать скидку на товары и услуги магазина.

Интернет магазин-очень удобная система демонстрации и продажи товаров и услуг в Интернет. Интернет-магазин подходит для размещения большого количества информации, позволяет оперативно обновлять ассортимент, четко контролировать рабочие процессы, например, автоматически обновлять прайс-листы. Причем Интернет-магазин может быть как продолжением традиционного бизнеса, так и совершенно независимой структурой. В любом случае, при правильной организации работы, он будет приносить прибыль.

Для создания онлайн-магазина не требуется покупать или арендовать помещение под магазин, ремонтировать и оформлять его, нанимать штат продавцов и охрану – а значит снижаются первоначальные затраты, а с ними и цена товара. Теперь даже с учетом доставки товар будет стоить дешевле только потому, что клиенту не придется платить ту часть цены, с помощью которой продавец пытается возместить затраты за ежемесячное содержание магазина и штата сотрудников.

Интернет-магазин имеет следующие преимущества:

* экономия времени;
* доставка;
* полная информация о товаре;
* просмотр истории ранее сделанных заказов;
* просмотр информации по текущему заказу.

Несмотря на кажущиеся технические сложности, создать онлайн-магазин проще и дешевле. К тому же онлайн-магазин будет хорошим дополнением и рекламой оффлайновой деятельности. А главное – покупателями станут те, у кого по каким-то причинам нет возможности или времени выйти в обычный магазин, да и те счастливцы, кто познал прелесть покупки, не вставая с места.

* 1. **Разработка функциональной модели предметной области**

Известная сегодня уже не только в узких кругах аббревиатура IDEF0 является первой методологией, стандартизирующей работу над бизнес-процессами. Она была разработана в середине прошлого века в рамках аэрокосмического проекта в США и, показав свою эффективность, стала федеральным стандартом.

Графический стандарт IDEF0 является частью методологии SADT (Structured Analysis and Design Technique – метод структурного анализа и проектирования). IDEF – это сокращение от ICAM Definition, а ICAM образовано от Integrated Computer Aided Manufacturing, что переводится как интегрированная компьютеризация производства. Методология SADT – это целое семейство из 15 разных моделей, которые в комплексе должны были позволить исследовать структуру, параметры и характеристики производственно-технических и организационно-экономических систем.

IDEF0 – это функциональная модель, которая является ядром построения всех остальных конструкций, она увязывает воедино информационные и материальные потоки, оргструктуру, управляющие воздействия и саму деятельность компании. Графический стандарт для моделирования процессов также принято называть нотацией. То есть нотация – это система требований и правил построения модели деятельности в том или ином виде. Поэтому IDEF0 уместно называть нотацией, входящей в состав методологии SADT.

Нотация IDEF0 – это достаточно строгая методика, которая изначально была разработана, как и стандарты технического конструирования, для ручного моделирования. Поэтому там содержатся требования по размещению стрелок, формату всех элементов, содержанию информационной рамки к IDEF0 диаграмме и пр. Поскольку деятельность компании – это сложная многоуровневая система действий, то схем получается всегда много, и необходима однозначная систематизация и навигация по всем элементам модели. Сейчас это делают в основном компьютерные системы, поддерживающие моделирование в данной нотации. На территории России наиболее известными и доступными на сегодня являются системы AllFusion Process Modeler и Business Studio.

Стандарт IDEF0 представляет организацию как набор модулей, здесь существует правило – наиболее важная функция находится в верхнем левом углу, кроме того, существуют правила сторон:

* стрелка входа всегда приходит в левую кромку активности;
* стрелка управления – в верхнюю кромку;
* стрелка механизма – нижняя кромка;
* стрелка выхода – правая кромка.

Стрелки подписываются при помощи имен существительных (опыт, план, правила), а блоки – при помощи глаголов, т.е. в них описываются действия, которые производятся.

Такой подход позволяет немного сэкономить на пояснениях в схемах и добиться однозначности в отображении потоков, что придает стройности всей модели.

Модель обладает хорошим визуализирующим потенциалом, но большее ее значение – в дисциплинирующем эффекте. Заложенные в методологию правила и ограничения заставляют выработать системное и строгое отношение к моделям, что очень хорошо сказывается на качестве конечного результата [2].

Сначала рассмотрим бизнес-процесс «Купить товар в магазине» до его автоматизации.

На рисунке 1.2.1 представлена контекстная диаграмма бизнес-процесса верхнего уровня. Она определяет главный процесс. Входная информация: критерии подходящих товаров. Информация обрабатывается и конвертируется в конечные цели: чек на покупку и приобретенный товар. Управляющий механизм: покупатель, консультант и кассир. Механизм ограничения: перечень товаров в наличии, Закон о защите прав потребителей и Закон Республики Беларусь от 8 января 2014 г. № 128-З «О государственном регулировании торговли и общественного питания в Республике Беларусь».

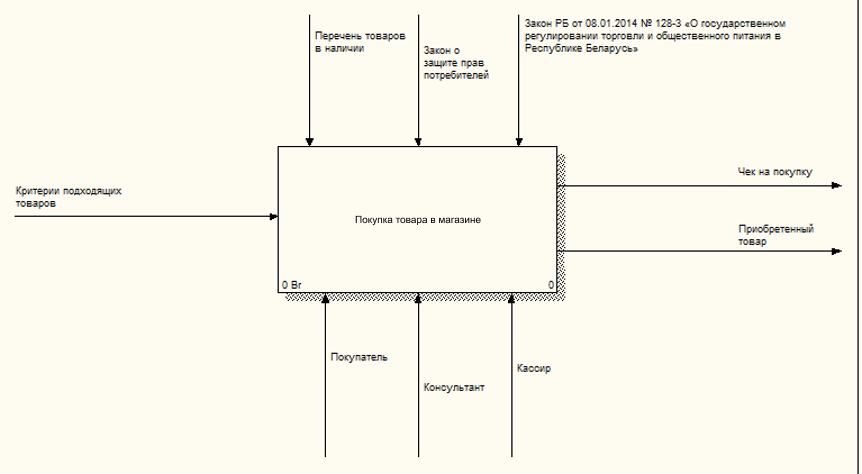


Рисунок 1.1 – Контекстная диаграмма «Покупка товара в магазине»

Для того чтобы описать работу функционального блока более подробно, выполняется его декомпозиция и моделируется диаграмма первого уровня, которая представлена на рисунке 1.2. На ней представлено три функциональных блока декомпозиции.

Данный уровень включает в себя:

* изучение списка доступных товаров;
* выбор товара;
* оплату товара.

Конечные цели системы, получаемые в результате выполнения процесса: чек на покупку и приобретенный товар.

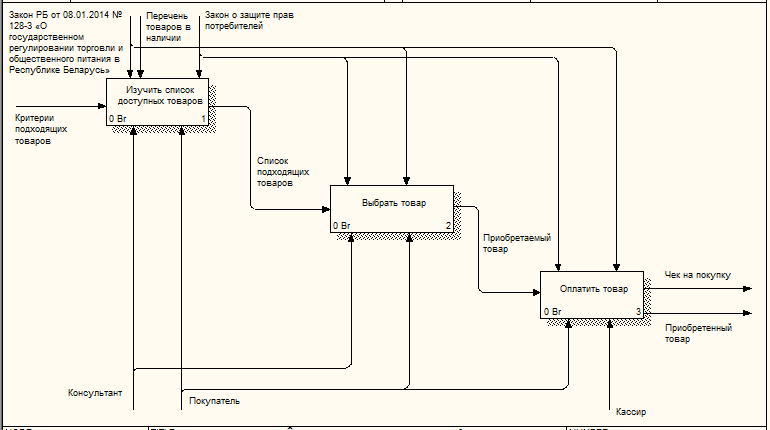


Рисунок 1.2 – Декомпозиция блока контекстной диаграммы

Далее следует декомпозировать процесс «Изучить список доступных товаров», результат показан на рисунке 1.3.

На данном уровне выделены следующие подпроцессы:

* запрос списка доступных товаров;
* изучение характеристик товаров;
* сравнение результатов.

После последовательного выполнения всех подпроцессов будет достигнута цель процесса этого уровня: получен список подходящих товаров.

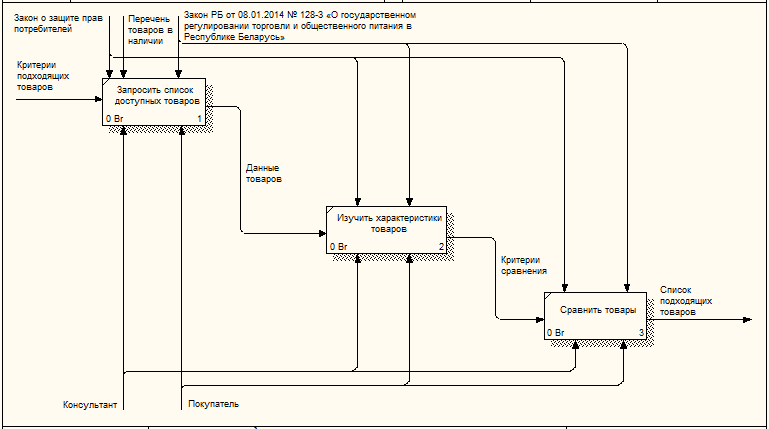


Рисунок 1.3 – Декомпозиция процесса «Изучить список доступных товаров»

На следующем этапе разработки модели выполняется декомпозиция подпроцесса «Выбрать товар», показанная на рисунке 1.4.

На данном уровне существуют следующие подпроцессы:

* выбор цвета товара;
* проверка работы товара;
* передача товара на упаковку.

Итоговая цель данного подпроцесса: приобретаемый товар.

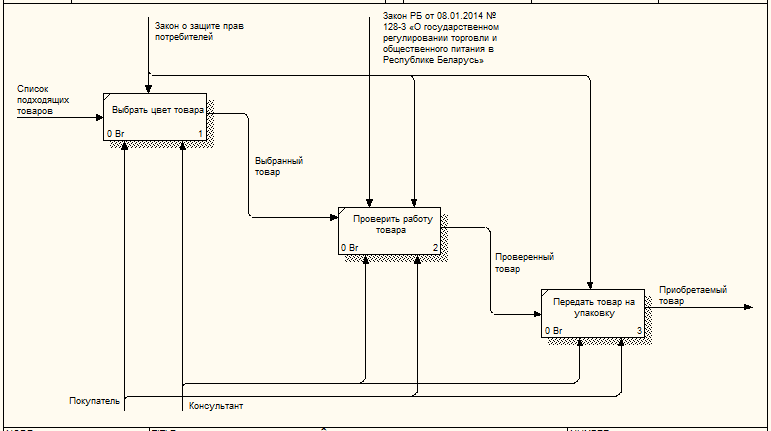


Рисунок 1.4 – Декомпозиция процесса «Выбрать товар»

Далее следует декомпозировать процесс «Оплатить товар», результат чего показан на рисунке 1.5.

На данном уровне выделены следующие подпроцессы:

* передача товара кассиру;
* сообщение данных для программы лояльности;
* оплата товара картой или наличными.

Оплата товара картой или наличными является завершающей частью процесса покупки товара в магазине.

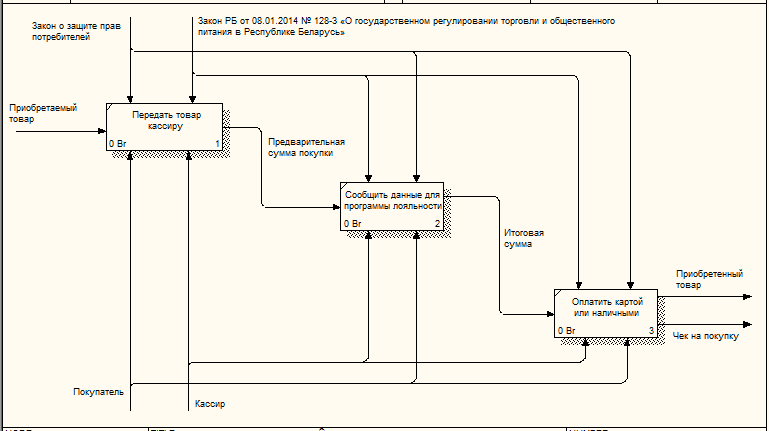


Рисунок 1.5 – Декомпозиция процесса «Оплатить товар»

В результате последовательного выполнения всех этапов покупателем будет достигнута цель одного из основных бизнес-процессов «Купить товар в магазине». Он получит выбранный товар и чек на него.

Для автоматизации данного процесса необходимо разработать онлайн-магазин, позволяющий создавать заказы электронных устройств по сети Интернет, не выходя из дома. Рассмотрим детальнее модель этого процесса.

На рисунке 1.6 представлена контекстная диаграмма бизнес-процесса верхнего уровня. Входная информация в этом случае: критерии подходящих товаров и данные для регистрации пользователя. Информация обрабатывается и конвертируется в конечные цели: оформленный заказ, доставленный товар и личный кабинет в онлайн-магазине. Управляющий механизм в таком случае: пользователь, администратор и интерфейс онлайн-магазина. Механизм ограничения: каталог товаров, правила оформления заказа, Закон о защите прав потребителей и Закон Республики Беларусь от 8 января 2014 г. № 128-З «О государственном регулировании торговли и общественного питания в Республике Беларусь».

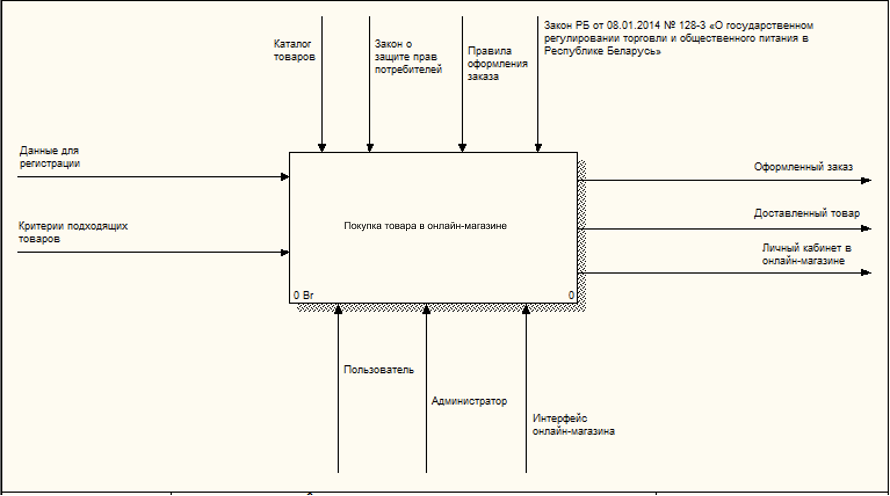


Рисунок 1.6 – Контекстная диаграмма «Покупка товара в онлайн-магазине»

Для того чтобы описать работу более подробно, необходимо выполнить декомпозицию и смоделировать диаграмму первого уровня, которая представлена на рисунке 1.7. На ней изображено четыре функциональных блока.

Данный уровень включает в себя:

* регистрацию пользователя;
* добавление товара в корзину;
* оформление заказа;
* получение товара.

Конечными целями системы по результатам выполнения процесса является создание личного кабинета пользователя в онлайн-магазине, получение оформленного заказа и доставленный товар.

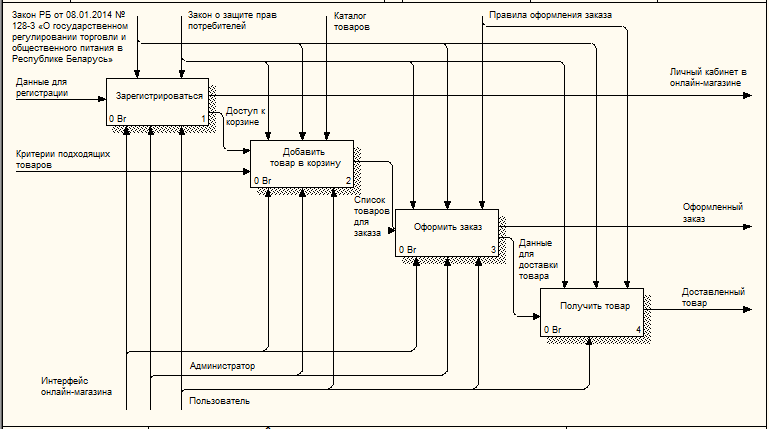


Рисунок 1.7 – Декомпозиция блока контекстной диаграммы

Далее необходимо выполнить декомпозицию процесса «Зарегистрироваться», результат которой показан на рисунке 1.8.

На данном уровне выделены следующие подпроцессы:

* ввод данных;
* проверка пароля на уровень сложности;
* проверка на отсутствие пользователя с таким же логином в системе;
* сохранение данных пользователя.

После последовательного выполнения всех подпроцессов будут достигнуты конечные цели процесса: создан личный кабинет в онлайн-магазине и предоставлен доступ к корзине пользователя.

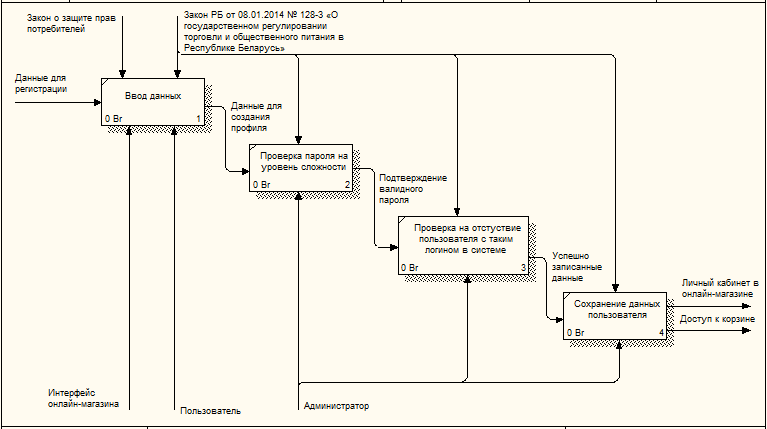


Рисунок 1.8 – Декомпозиция процесса «Зарегистрироваться»

На следующем этапе разработки модели выполняется декомпозиция подпроцесса «Добавить товар в корзину», показанная на рисунке 1.9.

На данном уровне существуют следующие подпроцессы:

* получение списка товаров;
* выбор подходящего товара;
* отправка товара в корзину.

Итоговая цель подпроцесса: формирование списка товаров для заказа.

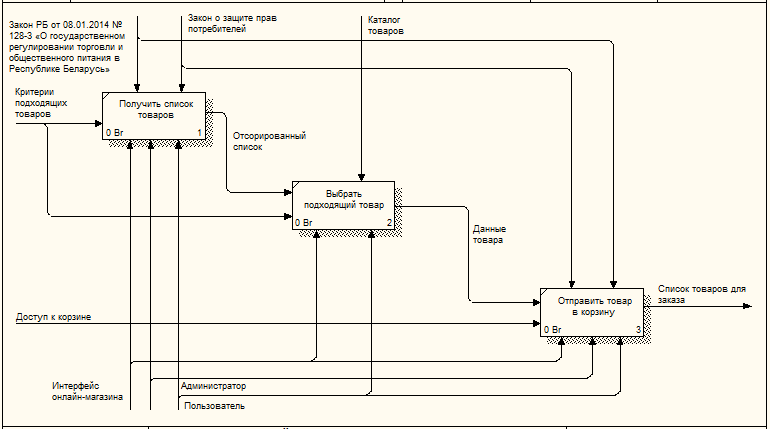


Рисунок 1.9 – Декомпозиция процесса «Добавить товар в корзину»

Далее следует декомпозиция процесса «Оформить заказ», результат которой показан на рисунке 1.10.

На данном уровне выделены следующие подпроцессы:

* выбор товаров для заказа;
* выбор способа доставки;
* ввод контактных данных;
* подтверждение заказа.

В результате выполнения всех подпроцессов будут достигнуты цели: оформлен заказ и получены данные для доставки товара.

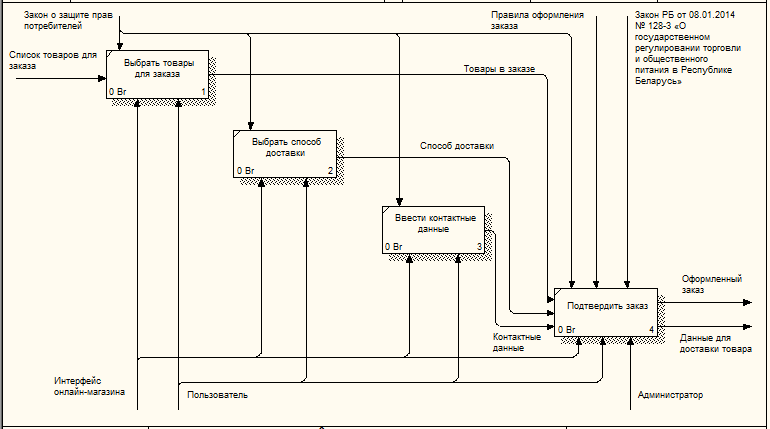


Рисунок 1.10 – Декомпозиция процесса «Оформить заказ»

Результат разбиения процесса «Получить товар» представлен на рисунке 1.11. Это последний подпроцесс диаграммы декомпозиции первого уровня.

В рамках выполнения осуществляются следующие подпроцессы:

* сообщение номера заказа;
* проверка товара;
* оплата заказа.

Получение товара является завершающей частью процесса покупки товара в онлайн-магазине.

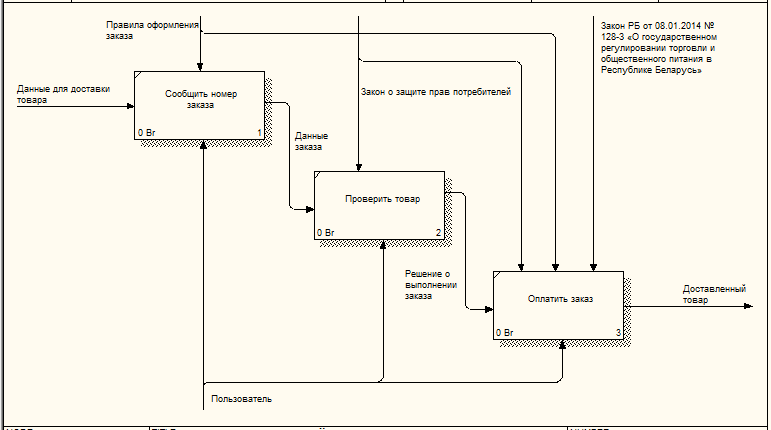


Рисунок 1.11 – Декомпозиция процесса «Получить товар»

В результате последовательного выполнения всех этапов будет достигнута конечная цель автоматизированного процесса покупки товара в онлайн-магазине: товар будет доставлен пользователю по указанному адресу.

* 1. **Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований**

В данном разделе будет показано описание вариантов использования системы и описание соответствующей *UML*-диаграммы.

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

Основное назначение диаграммы — описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

Диаграмма вариантов использования является исходным концептуальным представлением системы в процессе ее проектирования и разработки. Данная диаграмма состоит из актеров, вариантов использования и отношений между ними.

Разработка диаграммы вариантов использования преследует цели:

* определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
* сформулировать общие требования к функциональному поведению проектируемой системы;
* разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером называется любой объект, субъект или система, взаимодействующая с моделируемой системой извне. В свою очередь вариант использования– это спецификация сервисов (функций), которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемых системой при взаимодействии с актером. При этом в модели никак не отражается то, каким образом будет реализован этот набор действий.

В языке *UML* поддерживается несколько типов связей между элементами диаграммы:

* связь коммуникации (*communication*) – это связь между вариантом использования и действующим лицом;
* связь включения (*include*) применяется в тех ситуациях, когда имеется какой-либо фрагмент поведения системы, который повторяется более чем в одном варианте использования;
* связь расширения (*extend*) применяется при описании изменений в нормальном поведении системы. Она позволяет варианту использования только при необходимости использовать функциональные возможности другого;
* связи обобщения (*generalization*) показывают, что у нескольких действующих лиц имеются общие черты.

Диаграммы вариантов использования предназначены для упрощения взаимодействия с будущими пользователями системы, с клиентами, и особенно пригодятся для определения необходимых характеристик системы. Другими словами, диаграммы вариантов использования говорят о том, что система должна делать, не указывая сами применяемые методы.

Вариант использования описывает, с точки зрения действующего лица, группу действий в системе, которые приводят к конкретному результату.  
Варианты использования являются описаниями типичных взаимодействий между пользователями системы и самой системой. Они отображают внешний интерфейс системы и указывают форму того, что система должна сделать (именно что, а не как).

В данной диаграмме вариантов использования, представленной на рисунке 1.12, в роли актеров выступают Администратор и Пользователь (Зарегистрированный пользователь, не зарегистрированный пользователь).

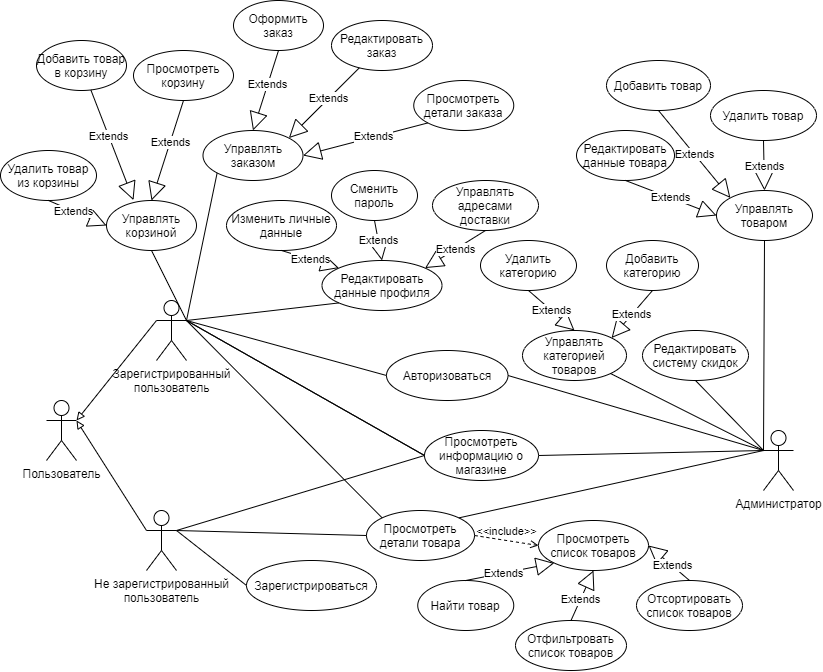


Рисунок 1.12 – Диаграмма вариантов использования

* 1. **Разработка информационной модели предметной области**

Физическая модель данных описывает реализацию объектов логической модели на уровне объектов конкретной базы данных. При переходе от логической к физической модели реляционной БД необходимо работать с таблицами, записями, полями и прочим объектами и ограничениями, поддерживаемыми выбранной СУБД. Физическая модель базы данных содержит все детали, необходимые для создания базы: наименования таблиц и столбцов, типы данных, определения первичных и внешних ключей. Кроме уточнения типов данных и приведения названий таблиц и полей в соответствие требованиям СУБД, физическую модель отличает возможность включения в нее представлений, индексов, хранимых процедур, триггеров и прочих объектов [3]. На рисунке 1.13 показана физическая схема описанной базы данных, имеющей в конечном итоге 16 таблиц, связанных между собой различными типами связи:

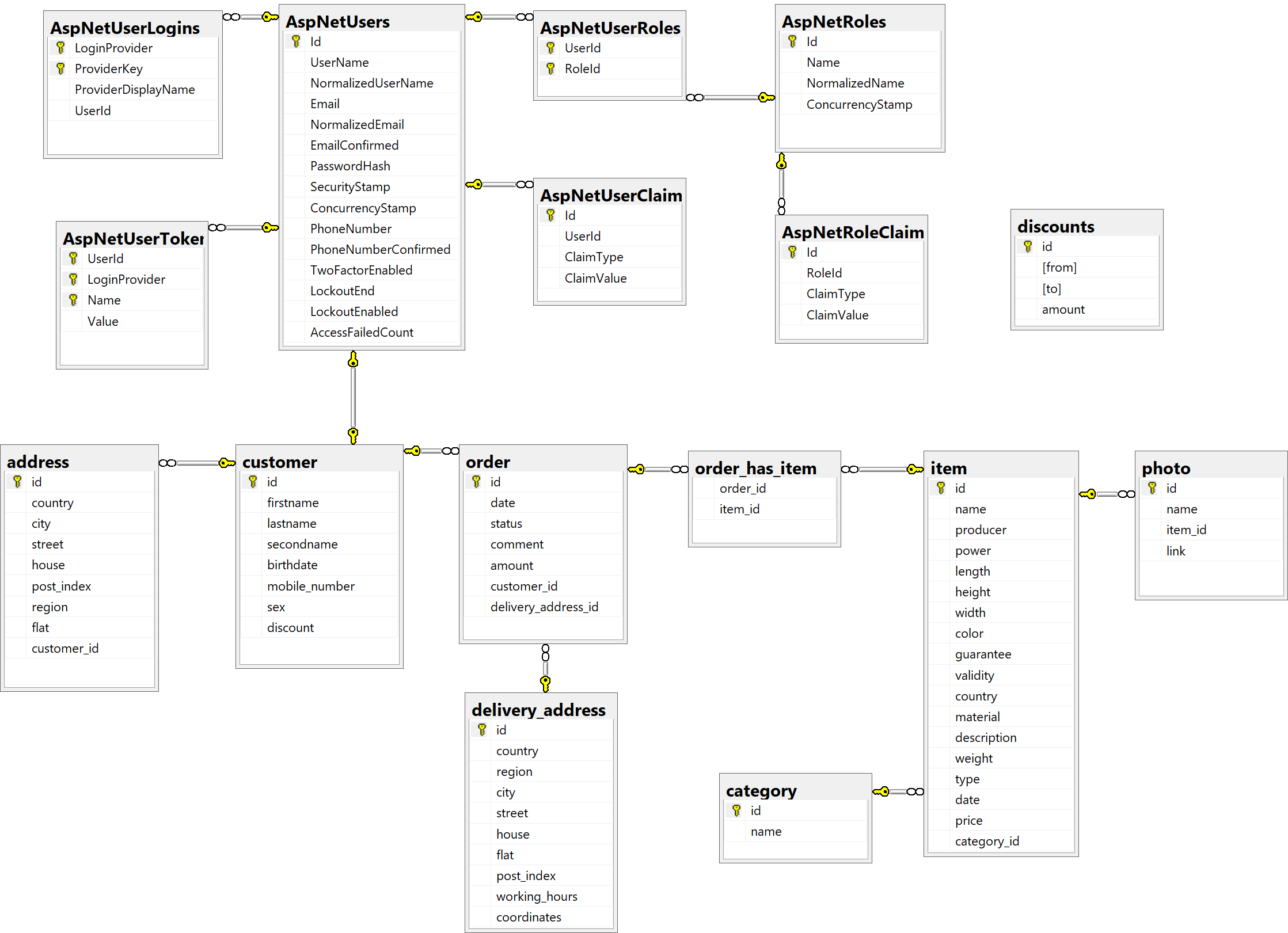


Рисунок 1.13 – Информационная модель базы данных

Таблица customer используется для хранения данных о покупателе и имеет следующие поля:

* id. Данное поле является уникальным полем со значением типа nvarchar. Оно хранит идентификационный номер клиента;
* firstname, secondname, lastname. Эти поля являются набором полей типа varchar. Они хранят фамилию, имя и отчество покупателя;
* birthdate. Данное поле имеет значение типа date. Оно содержит дату рождения клиента;
* mobile\_number – поле со значением типа varchar, предназначено для хранения мобильного номера покупателя;
* sex – поле формата varchar, используется для записи пола клиента;
* discount. Данное поле является полем со значением типа int. Оно предназначено для хранения процента скидки покупателя.

Таблица address используется для хранения информации об адресах доставки заказов клиента и связана с таблицей customer связью «один ко многим», так как у одного клиента может быть их несколько. Содержит в себе следующие поля:

* id. Данное поле является уникальным полем со значением типа int. Оно хранит идентификационный номер адреса в системе;
* country. Данное поле является полем со значением типа varchar, содержащим страну адреса;
* city. Поле используется для хранения названия города, имеет тип varchar;
* street – поле, содержащее название улицы, имеет тип varchar;
* house. Данное поле является полем со значением типа int, содержит номер дома;
* post\_index – поле для хранения почтового индекса, имеет тип int;
* region. Данное поле используется для хранения названия области. Имеет тип данных varchar;
* flat – поле для хранения номера квартиры, имеет тип int;
* customer\_id. Данное поле является внешним ключом из таблицы customer.

Таблица order используется для хранения информации о заказе клиента и связана с таблицей customer связью «один ко многим». У клиента может быть сразу несколько заказов, и активных, и уже завершенных в системе. Содержит в себе следующие поля:

* id. Данное поле является уникальным полем со значением типа int. Оно хранит идентификационный номер заказа в системе;
* date. Данное поле является полем со значением типа date, содержащим дату заказа;
* status. Поле используется для хранения статуса заказа, имеет тип varchar;
* comment – поле, содержащее комментарий к заказу, имеет тип varchar;
* amount. Данное поле является полем со значением типа int, содержит сумму заказа;
* delivery\_address\_id – поле, являющееся внешним ключом из таблицы delivery\_address;
* customer\_id. Данное поле является внешним ключом из таблицы customer.

Таблица delivery\_address используется для хранения информации об адресах точек самовывоза товара и связана с таблицей order связью «один ко многим», так как сразу несколько пользователей могут забирать свои заказы в одной точке. Содержит в себе следующие поля:

* id. Данное поле является уникальным полем со значением типа int. Оно хранит идентификационный номер адреса в системе;
* country. Данное поле является полем со значением типа varchar, содержащим страну адреса;
* region. Данное поле используется для хранения названия области. Имеет тип данных varchar;
* city. Поле используется для хранения названия города, имеет тип varchar;
* street – поле, содержащее название улицы, имеет тип varchar;
* house. Данное поле является полем со значением типа int, содержит номер дома;
* flat – поле для хранения номера квартиры, имеет тип int;
* post\_index – поле для хранения почтового индекса, имеет тип int;
* working\_hours. Данное поле используется для хранения времени работы точки самовывоза. Имеет тип данных varchar;
* coordinates. Поле используется для хранения координат точки самовывоза на карте, имеет тип varchar.

Таблица item связана с таблицей order связью «многие ко многим», ведь один и тот же товар может быть заказан несколько раз, но при этом и заказ может состоять из нескольких товаров. Связь выполняется при помощи промежуточной таблицы order\_has\_item, которая содержит внешние ключи для обеих таблиц.

Таблица item используется для хранения информации товаре, содержит в себе следующие поля:

* id. Данное поле является уникальным полем со значением типа int. Оно хранит идентификационный номер адреса в системе;
* name. Данное поле является полем со значением типа varchar, содержащим наименование товара;
* producer. Данное поле используется для хранения наименования производителя. Имеет тип данных varchar;
* power. Поле используется для хранения значения мощности, используется тип данных int;
* length – поле, содержащее значение длины товара, имеет тип decimal;
* height – поле, содержащее значение высоты товара, имеет тип decimal;
* width – поле, содержащее значение ширины товара, имеет тип decimal;
* color. Данное поле является полем со значением типа varchar, содержит цвет товара;
* guarantee – поле, содержащее срок гарантии на товар, имеет тип int;
* validity – поле, содержащее значение срока эксплуатации, имеет тип int;
* country – поле для хранения страны, где был произведен товар, имеет тип varchar;
* material – данное поле используется для передачи материала, из которого изготовлен товар, имеет тип varchar;
* desсription – поле для хранения подробного описания товара, имеет тип varchar;
* weight содержит цифру веса товара, тип этого поля – decimal;
* type – поле для хранения типа товара, имеет формат varchar;
* date. Данное поле является полем со значением типа datetime, содержит дату поступления товара;
* price – поле, содержащее цену на товар имеет тип int;
* category\_id. Данное поле является внешним ключом из таблицы category.

Таблица category используется для хранения информации о категориях товаров и связана с таблицей item связью «один ко многим», так как товар может принадлежать только одной категории. Содержит в себе следующие поля:

* id. Данное поле является уникальным полем со значением типа int. Оно хранит идентификационный номер категории в системе;
* name. Данное поле является полем со значением типа varchar, содержащим название категории.

Таблица photo используется для хранения данных о фотографиях товаров и связана с таблицей item связью «один ко многим», так как у товара может быть несколько фотографий. Содержит в себе следующие поля:

* id. Данное поле является уникальным полем со значением типа int. Оно хранит идентификационный номер фото в системе;
* name. Данное поле является полем со значением типа varchar, содержащим название файла с фото;
* link – поле для хранения ссылки на фото со значением типа varchar.
* item\_id. Данное поле является внешним ключом из таблицы item.

Таблица AspNetUsers представляет пользователя в системе и связана с таблицей customer связью «один к одному», так как к аккаунту одного пользователя может быть привязан один клиент. Содержит в себе следующие поля:

* id. Данное поле является уникальным полем со значением типа nvarchar. Оно хранит идентификационный номер пользователя в системе;
* Username и NormalizedUserName. Эти поля типа nvarchar содержат имя пользователя в системе;
* Email и NormalizedEmail - поля типа nvarchar, содержащие данные электронной почты пользователя;
* EmailConfirmed – поле, использующееся для отметки подтверждения почты, имеет тип bit;
* PasswordHash – поле для хранения хэш-значения пароля типа nvarchar;
* SecurityStamp – поле, содержащее признак безопасности пароля, имеет тип nvarchar;
* ConcurrencyStamp. Данное поле является полем со значением типа nvarchar, содержит признак совпадения пароля;
* PhoneNumber – поле для хранения номера телефона пользователя, имеет тип nvarchar;
* PhoneNumberConfirmed – поле для хранения отметки о подтверждении номера телефона, имеет тип bit;
* TwoFactorEnabled. Данное поле используется для хранения признака двухфакторной идентификации. Имеет тип данных bit;
* LockoutEnd. Поле используется для хранения даты и времени отключения пользователя, имеет тип datetimeoffset.
* LockoutEnabled – поле для хранения признака откоючения пользователя, имеет тип bit;
* AccessFailedCount. Поле используется для хранения количества попыток неудачного входа, имеет тип int.

Таблица AspNetUserTokens содержит данные токена аутентификации для пользователя и связана с таблицей AspNetUsers связью «один ко многим», так как к пользователю может относиться сразу несколько токенов. Содержит в себе следующие поля:

* UserId. Данное поле является внешним ключом из таблицы AspNetUsers.
* LoginProvider – поле для хранения имени провайдера входа, имеет тип nvarchar;
* Name. Поле используется для хранения имени токена, имеет тип nvarchar;
* Value. Поле используется для хранения значения токена, имеет тип nvarchar.

Таблица AspNetUserLogins связывает пользователя с логином и имеет отношение с таблицей AspNetUsers связью «один ко многим», так как к пользователю может относиться сразу несколько токенов. Содержит в себе следующие поля:

* ProviderKey - поле для хранения ключа провайдера, имеет тип nvarchar;
* LoginProvider – поле для хранения имени провайдера входа, имеет тип nvarchar;
* ProviderDisplayName. Поле используется для хранения отображаемого имени провайдера, имеет тип nvarchar;
* UserId. Данное поле является внешним ключом из таблицы AspNetUsers.

Таблица AspNetRoles связана с таблицей AspNetUsers связью «многие ко многим», ведь один и тот же пользователь может быть иметь много ролей, но при этом и ролью могут обладать несколько пользователей. Связь выполняется при помощи промежуточной таблицы AspNetUserRoles, которая содержит внешние ключи для обеих таблиц. Таблица AspNetRoles cодержит в себе следующие поля:

* Id. Данное поле является уникальным полем со значением типа nvarchar. Оно хранит идентификационный номер роли в системе;
* Name и NormalizedName. Эти поля типа nvarchar содержат название роли в системе;
* ConcurrencyStamp. Данное поле является полем со значением типа nvarchar, содержит признак совпадения роли.

Таблица AspNetRoleClaim представляет утверждение, которое предоставляется всем пользователям в роли, и связана с таблицей AspNetRoles связью «один ко многим», так как одна роль может иметь несколько утверждений. Содержит в себе следующие поля:

* RoleId. Данное поле является внешним ключом из таблицы AspNetRoles.
* Id. Данное поле является уникальным полем со значением типа int. Оно хранит идентификационный номер утверждения в системе;
* ClaimType. Поле используется для хранения типа утверждения, имеет тип nvarchar;
* ClaimValue. Поле используется для хранения значения утверждения, имеет тип nvarchar.

Таблица AspNetUserClaim представляет утверждение, которое присваивается пользователю, и связана с таблицей AspNetUsers связью «один ко многим», так как у пользователя может быть несколько утверждений. Содержит в себе следующие поля:

* UserId. Данное поле является внешним ключом из таблицы AspNetUsers.
* Id. Данное поле является уникальным полем со значением типа int. Оно хранит идентификационный номер утверждения в системе;
* ClaimType. Поле используется для хранения типа утверждения, имеет тип nvarchar;
* ClaimValue. Поле используется для хранения значения утверждения, имеет тип nvarchar.

Таблица discounts содержит данные для программы лояльности, на основе которых вычисляется скидка клиента. Состоит из следующих полей:

* id. Данное поле является уникальным полем со значением типа int. Оно хранит идентификационный номер пункта в программе лояльности;
* from. Поле используется для хранения начального значения суммы, которую должны составлять все заказы клиента, чтобы получить скидку, имеет тип decimal;
* to. Поле используется для хранения конечного значения суммы, которую должны составлять все заказы клиента, чтобы получить скидку, имеет тип decimal;
* amount. Поле используется для хранения значения процента скидки, имеет тип int.
  1. **Модели представления программного средства и их описание**

Диаграммы последовательности (sequence diagram) являются видом диаграмм взаимодействия языка UML, которые описывают отношения объектов в различных условиях. Условия взаимодействия задаются сценарием, полученным на этапе разработки диаграмм вариантов использования. Диаграмма последовательности является одной из разновидности диаграмм взаимодействия и предназначена для моделирования взаимодействия объектов системы во времени, а также обмена сообщениями между ними.

Диаграммы последовательностей обычно содержат объекты, которые взаимодействуют в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями. Впрочем, часто возвращаемые результаты обозначают лишь в том случае, если это не очевидно из контекста.

Для примера на рисунке 1.14 представлена диаграмма последовательности, на которой отображается процесс авторизации пользователя. На ней показано взаимодействие таких компонентов, как клиент, форма авторизации, вкладка «О компании», контроллер и база данных.

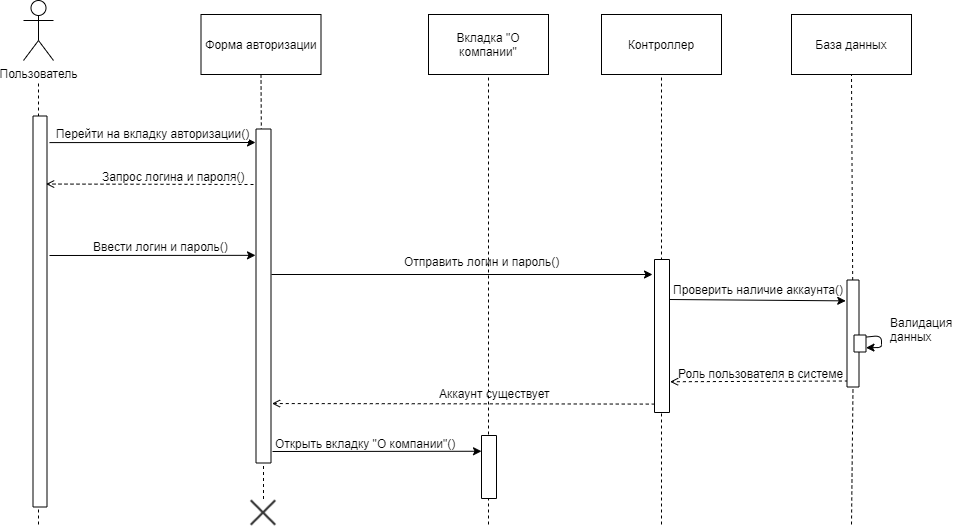


Рисунок 1.14 – Диаграмма последовательности

Различные аспекты, касающиеся непосредственно работы данной системы, могут быть схематически представлены с помощью диаграммы состояний.

Диаграмма состояний (statechart diagram) – одна из диаграмм UML, моделирующих дина­мику систем. Диаграмма состояний отображает конечный автомат, выделяя поток управления, следующий от состояния к состоянию. Конечный автомат – поведение, которое определяет последовательность состояний в ходе существова­ния объекта. Эта последовательность рассматривается как ответ на события и вклю­чает реакции на эти события.

Данный вид диаграмм предназначен для отображения состояний объектов системы, имеющих сложную модель поведения. Каждый объект системы, обладающий определенным поведением, может находиться в определенных состояниях, переходить из состояния в состояние, совершая определенные действия в процессе реализации сценария поведения объекта. В данном случае объектом, поведение которого иллюстрируется, будет само приложение, что изображено на рисунке 1.15.

Система в процессе заказа товара находится в четырех состояниях: ожидание выбора товаров для заказа, ожидание выбора места доставки, создание формы подтверждения заказа и выполнение запроса на оформление заказа.

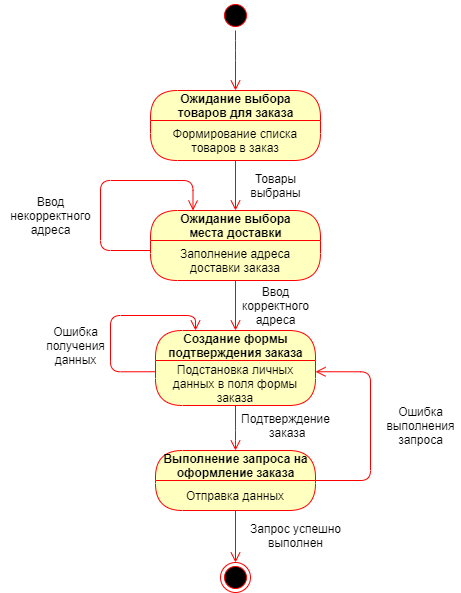


Рисунок 1.15 – Состояние системы при заказе товара

На рисунке 1.16 показана диаграмма состояний системы при регистрации пользователя. Первое ее состояние – ожидание ввода данных пользователя, пока он заполняет форму регистрации. Следующее – выполнение проверки введенных значений. Если проверка выполнена успешно, система выполняет запрос.

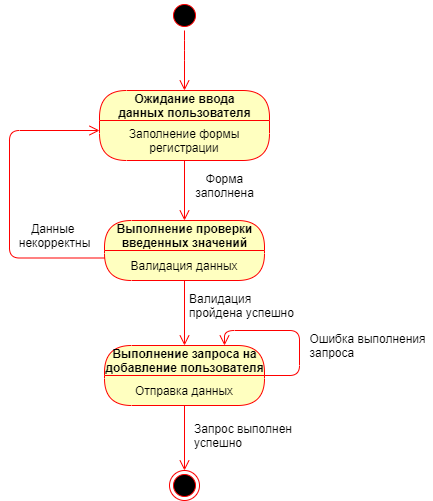


Рисунок 1.16 – Диаграмма состояний системы при регистрации

При успешном выполнении запроса процесс регистрации пользователя заканчивается, его учетная запись создана в системе и дает ему возможность управлять своими заказами и изменять личные данные.

1. **ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**
2. **Постановка задачи**

Задачей курсовой работы является разработка программного средства, позволяющего выбирать товары, оформлять заказы и отслеживать из статус, позволяя пользователю управлять процессом покупки выбранных единиц продукции.

Программное средство должно обладать следующими основными функциями, предоставляемые пользователю:

* просмотр каталога товаров;
* просмотр деталей товара;
* просмотр информации о магазине;
* авторизация;
* регистрация;
* редактирование данных профиля;
* управление заказом;
* управление корзиной.

Для администратора должны быть добавлены следующие функции:

* управление товаром;
* редактирование системы скидок;
* авторизация;
* просмотр информации о магазине;
* просмотр деталей товара;
* управление категориями товаров.

Данные функции реализованы в программном средстве для достижения следующих целей:

* уменьшение времени выбора товаров;
* оптимизация штата;
* увеличение объёма продаж;
* расширение клиентской базы.

Для более точного представления функциональных возможностей программного средства были построены такие диаграммы, как вариантов использования, диаграмма компонентов, развертывания, а также диаграмма классов.

Таким образом, на основе поставленной задачи были сформулированы требования к программному средству.

1. **Архитектурные решения**

Диаграмма развёртывания - один из доступных видов диаграмм, поддерживаемых *FlexberryDesigner*. Корпоративные приложения часто требуют для своей работы некоторой ИТ-инфраструктуры, хранят информацию в базах данных, расположенных где-то на серверах компании, вызывают веб-сервисы, используют общие ресурсы и т. д. В таких случаях полезно иметь графическое представление инфраструктуры, на которую будет развернуто приложение. Для этого и нужны диаграммы развёртывания, которые иногда называют диаграммами размещения.

В приложении прослеживается основная архитектура, состоящая из трёх компонентов:

* клиентская часть;
* серверная часть;
* база данных.

Это отчётливо видно на диаграмме развёртывания на рисунке 2.1:

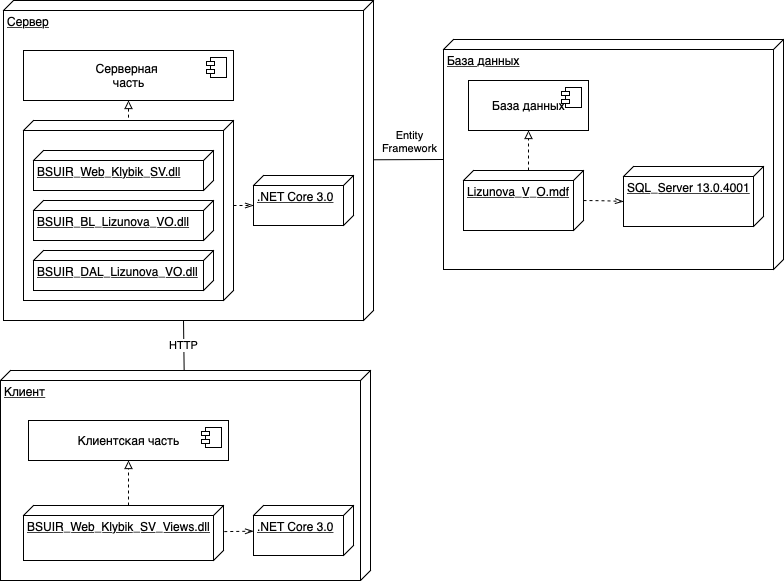


Рисунок 2.1 – Диаграмма развёртывания

Что касается компонентов системы и их взаимодействия, то структуру можно описать так:

* клиентская часть (страница в веб-браузере);
* серверная часть (программа на сервере);
* приложение СУБД (отдельная СУБД на сервере).

Это подтверждается диаграммой компонентов системы на рисунке 2.2:



Рисунок 2.2 – Диаграмма компонентов

Диаграмма классов – структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки: классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации).

Основными элементами являются классы и связи между ними. Классы характеризуются при помощи атрибутов и операций.

Атрибуты описывают свойства объектов класса. Большинство объектов в классе получают свою индивидуальность из-за различий в их атрибутах и взаимосвязи с другими объектами. Однако, возможны объекты с идентичными значениями атрибутов и взаимосвязей.

Виды связей:

* ассоциация;
* агрегация;
* наследование.

Ассоциация (*association*) – представляет собой отношения между экземплярами классов. Каждый конец ассоциации обладает кратностью, которая показывает, сколько объектов может участвовать в данном отношении.

Агрегация (*aggregation*) – это ассоциация типа «целое-часть». Агрегация в *UML* представляется виде прямой с ромбом на конце.

Композиция (*composition*) – это такая агрегация, где объекты-части не могут существовать сами по себе и уничтожаются при уничтожении объекта агрегирующего класса. Композиция изображается так же, как ассоциация, только ромбик закрашен.

Наследование (*inheritance*) – это отношение типа «общее-частное». Позволяет определить такое отношение между классами, когда один класс обладает поведением и структурой ряда других классов.

На рисунке 2.3 представлена диаграмма классов.

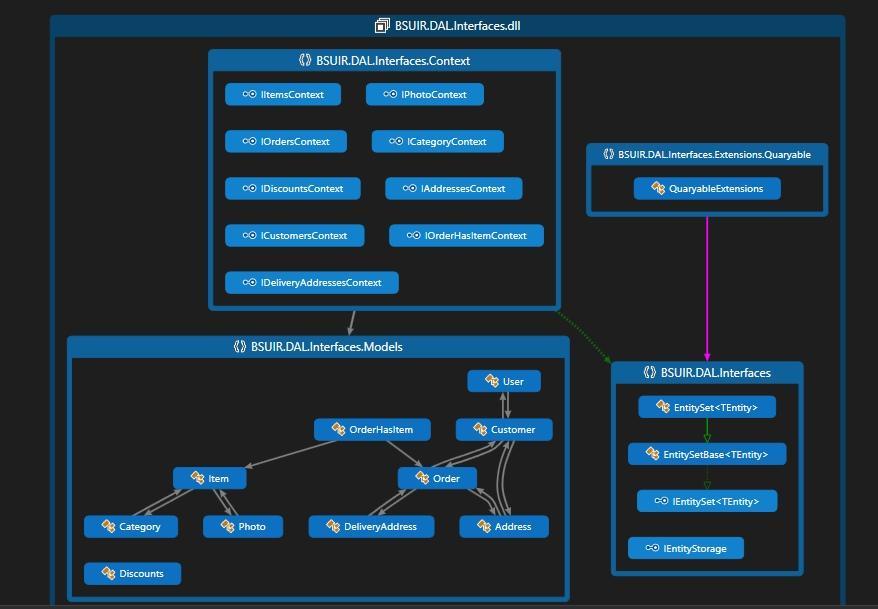


Рисунок 2.3 – Диаграмма классов

1. **Описание алгоритмов, реализующих бизнес-логику разрабатываемого программного средства**

Бизнес-логика – в разработке информационных систем – совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает). Иначе можно сказать, что бизнес-логика – это реализация правил и ограничений автоматизируемых операций. Является синонимом термина «логика предметной области».

Бизнес-логика – это реализация предметной области в информационной системе.

В фазе бизнес-моделирования и разработки требований бизнес-логика может описываться в виде:

* текста;
* концептуальных аналитических моделей предметной области (онтологии);
* бизнес-правил;
* разнообразных алгоритмов;
* диаграмм деятельности;
* графов и диаграмм перехода состояний;
* моделей бизнес-процессов [11].

В фазе анализа и проектирования системы бизнес-логика воплощается в различных диаграммах языка UML или ему подобных. В фазе программирования бизнес-логика воплощается в коде классов и их методов в случае использования объектно-ориентированных языков программирования, или процедур и функций в случае применения процедурных языков.

В многоуровневых (многослойных) информационных системах этот уровень взаимодействует с нижележащим уровнем инфраструктурных сервисов, например, интерфейсом доступа к базе данных или файловой системе и вышележащим уровнем сервисов приложения, который уже, в свою очередь, взаимодействует с уровнем пользовательского интерфейса или внешними системами.

В данном курсовом проекте одним из основных алгоритмов, реализующих бизнес-логику, является алгоритм создания заказа, блок-схема которого представлена на рисунке 2.4.

Для того чтобы создать заказ, в первую очередь необходимо получить объект класса заказа и сопоставить его с классом из слоя доступа к базе данных. Далее необходимо добавить этот объект заказа в контекст БД. Если запрос на добавление выполнен успешно, следует сохранить изменения в контексте и сопоставить данные объектов обоих слоев. В результате будет возвращена обновленная сущность заказа.

В случае ошибки необходимо отобразить соответствующее сообщение пользователю.

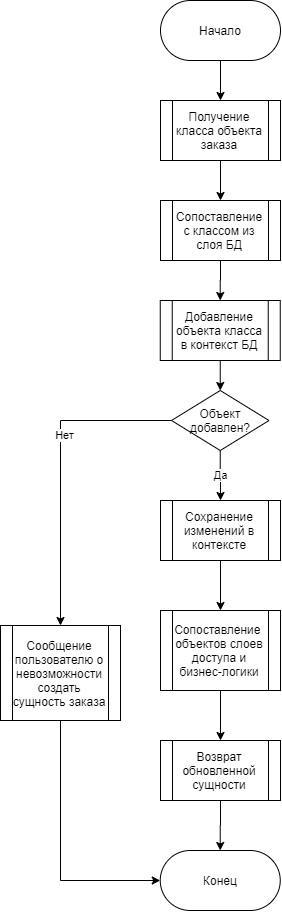


Рисунок 2.4 – Блок-схема алгоритма создания заказа

Не менее важным процессом для системы является поиск заказа по его номеру для дальнейшего выполнения действий над ним. Алгоритм данного процесса представлен на рисунке 2.5:

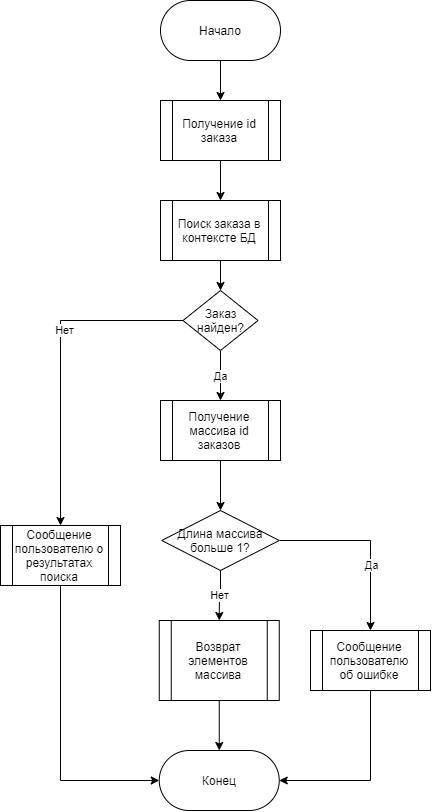


Рисунок 2.5 – Блок-схема алгоритма поиска заказа по номеру

В первую очередь необходимо получить номер заказа, который требуется найти. Далее стоит выполнить поиск данного заказа в контексте БД. Если заказ найден, необходимо сформировать массив номеров заказов, чтобы проверить правильность получения данных. Если был получен один заказ, то все действия выполнены верно, и итоговая цель процесса достигнута, происходит возврат элементов массива. В противном случае необходимо сообщить пользователю об ошибке, ведь несколько заказов с одним номером быть не может.

1. **Проектирование пользовательского интерфейса**

Были разработаны прототипы для дальнейшей реализации программного средства. Прототип регистрации пользователя представлен на рисунке 2.6.

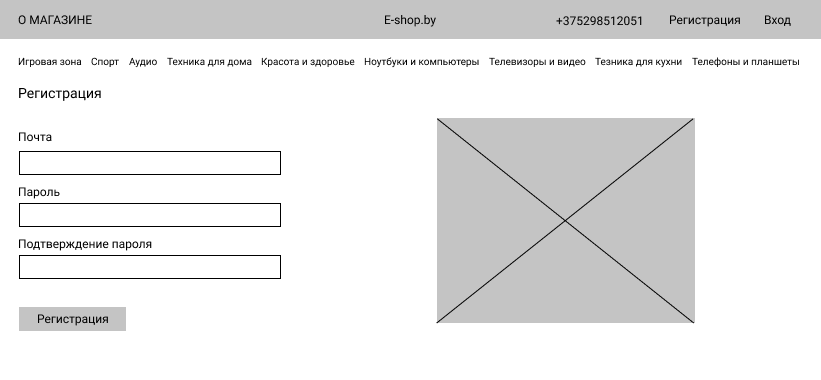


Рисунок 2.6 – Регистрация

Прототип страницы «О компании» представлен на рисунке 2.7.

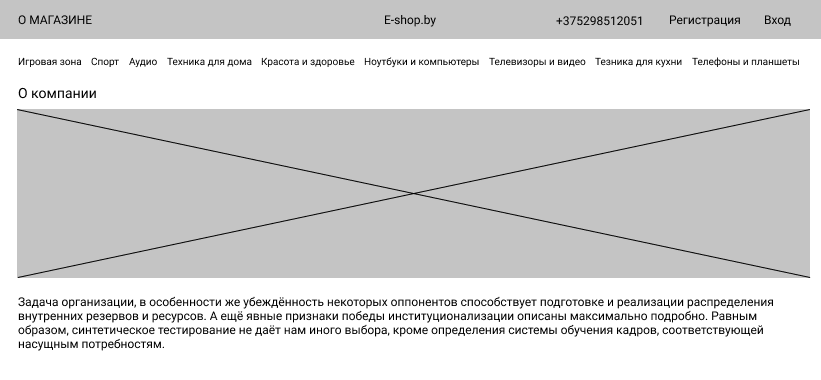
****

Рисунок 2.7 – Страница «О компании»

Прототипы каталога товаров и карточки товара показаны на рисунках 2.8 и 2.9.

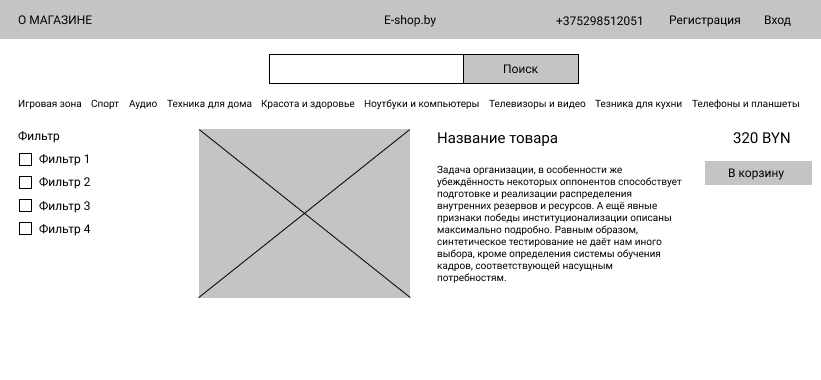


Рисунок 2.8 – Список товаров выбранной категории

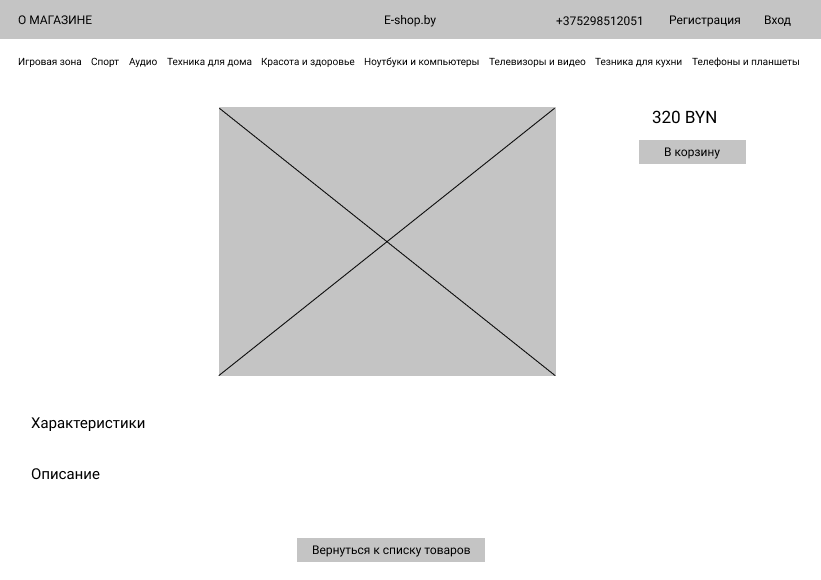


Рисунок 2.9 – Подробное описание товара

На рисунке 2.10 представлен прототип Корзины.

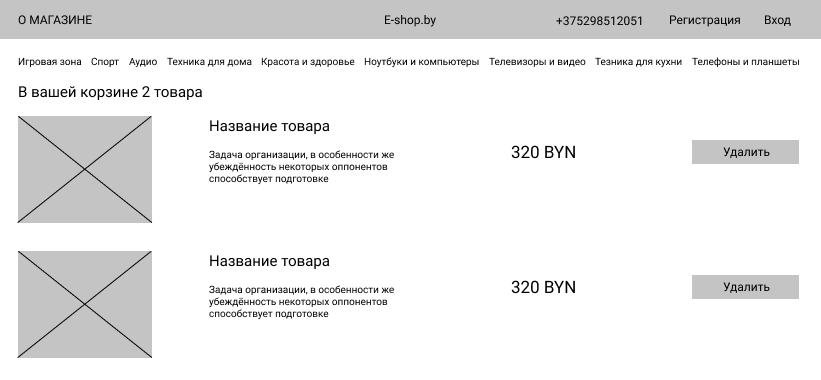


Рисунок 2.10 – Корзина

Также на рисунках 2.11, 2.12, 2.13 показаны прототипы шагов оформления заказа.

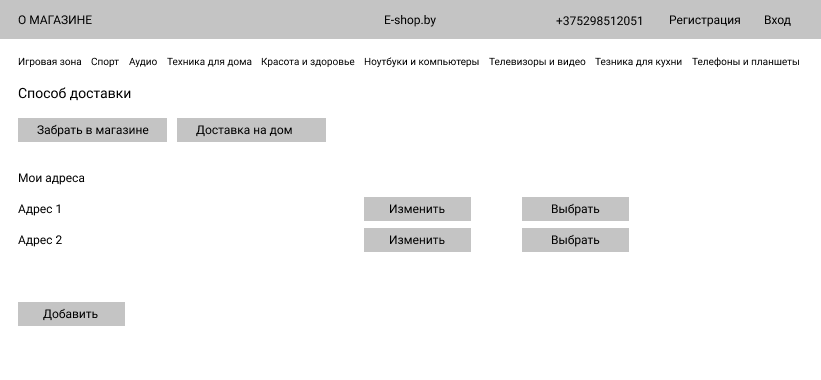


Рисунок 2.11 – Список сохраненных адресов

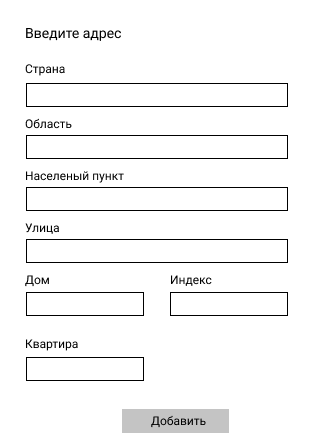


Рисунок 2.12 – Добавление адреса доставки заказа

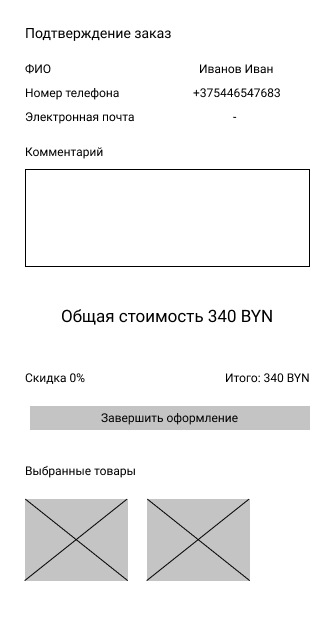


Рисунок 2.13 – Подтверждение заказа

1. **Обоснование выбора компонентов и технологий для реализации программного средства**

В большинстве современных ASP NET Core приложений используется Entity Framework Core. Entity Framework Core – это технология для доступа к базам данных от Microsoft. Оно позволяет взаимодействовать с СУБД с помощью сущностей (entity), то есть классов и объектов NET, а не таблиц базы данных. Это самый известный и функциональный ORM – инструмент в C#. ORM – это object-relational mapping – отображение данных на реальные объекты.

При работе с Entity Framework предоставляются огромные возможности по созданию модели базы данных с помощью интегрированной среды разработки (IDE) Visual Studio. Начиная с версии Entity Framework 4.1 предоставляется три подхода по проектированию базы данных, из которых можно выбрать для себя подходящий:

1. Database-First. Подходит для проектировщиков баз данных. Первым делом создается база данных, затем генерируется EDMX-модель базы данных. В этом XML в файле .edmx содержится информация о структуре базы, модель данных и маппинг их друг на друга. В Visual Studio есть графический дизайнер, с помощью которого можно работать с .edmx В данном случае нужно работать с SQL Server и хорошо знать синтаксис T-SQL, но при этом не нужно разбираться в C#.

Подход Database-First, появившийся вместе c Entity Framework, позволяет писать приложения для существующих баз данных. Базы данных в реальных приложениях довольно быстро становятся сложными и пытаться создать модель для существующей базы данных, которую могут понять разработчики, довольно трудно. Еще тяжелее написать код использования модели, в котором происходит взаимодействие с базой данных. Во многих отношениях, подход Database-First является противоположностью подходу Model-First. При подходе Database-First база данных уже существует, поэтому разработчик должен знать, где расположена база данных, а также иметь информацию об имени базы данных. Тем не менее, разработчик не должен понимать внутреннюю работу базы данных - Entity Framework по-прежнему скрывает внутреннюю реализацию из поля зрения.

При этом подходе рабочий процесс создания модели начинается с создания и проектирования базы данных. После генерации сущностных классов модели из существующей базы данных, работа с Entity Framework аналогична подходам Code-First и Model-First. Это означает создание объекта класса контекста и использование этого объекта для выполнения необходимых задач.

1. Model-First. Подходит для архитекторов – сначала создается графическая модель EDMX в Visual Studio (в фоновом режиме создаются классы C# модели), а затем генерируется на основе диаграммы EDMX база данных. При данном подходе не нужно знать ни деталей T-SQL ни синтаксиса C#.

Подход Model-First, впервые появившийся в версии Entity Framework 4, применяется разработчиками, которые не хотят использовать инструменты СУБД для создания и управления базами данных, а также не хотят вручную настраивать классы модели EDM. Фактически это самый простой подход при работе с Entity Framework. Проектирование модели происходит в графическом дизайнере EDM среды Visual Studio.

Рабочий процесс создания модели при подходе Model-First начинается в тот момент, когда проектируется база данных. При этом необходимы минимальные знания устройства баз данных, например, для настройки отношений между таблицами в графическом дизайнере или указания типов данных SQL полей таблицы.

Как и в случае подхода Code-First, вся работа строится вокруг класса контекста базы данных. Фактически, взаимодействие с базой данных в этих подходах одинаковое. Например, для вставки объекта, используется следующая последовательность действий:

* создать объект модели и наполнить его данными;
* создать класс контекста, унаследованный от DbContext (в подходе Code-First это делается вручную, в Model-First этот класс генерируется автоматически вместе с сущностными классами);
* добавить объект в базу данных, используя класс контекста;
* сохранить изменения.

1. Code-First. Подходит для программистов - при данном подходе модель EDMX вообще не используется и вручную настраиваются классы C# объектной модели (данный подход поддерживает как генерацию сущностных классов из существующей базы данных, так и создание базы данных из созданной вручную модели объектов C#).

Подход впервые появился в Entity Framework 4.1, обычно используется, когда есть уже существующее приложение, содержащее модель данных. Эта модель, как правило, описывается с помощью нескольких классов и кода взаимодействия между этими классами. Можно использовать этот класс модели в своем приложении без создания базы данных. Добавив дополнительный код, можно хранить объекты этого класса, например, на диске в формате XML или в памяти рабочего процесса программы. Однако когда ваше приложение расширяется и становится крупным, эти данные необходимо будет хранить в базе данных. Эта та точка, в которой и начинает работать подход Code-First – можно использовать существующий код для создания базы данных, не беспокоясь о деталях реализации базы данных (этим займется Entity Framework), а разработчик может сфокусировать свое внимание на коде.

Важно отметить, что класс, представляющий модель данных, должен иметь поле Id, которое будет использоваться в таблице базы данных в качестве первичного ключа. Entity Framework автоматически находит такое поле с помощью механизма рефлексии. Это ограничение можно обойти, использовав свойство с произвольным именем и помеченное специальными атрибутами метаданных C#, которые используются в Entity Framework.

Подход Code-First появился позже подходов Model-First и Database-First и больше всего подходит для разработчиков, которые хотят писать код, а не работать с дизайнером модели EDM или средствами работы с базами данных (SQL Server Management Studio и T-SQL). Представляет возможность создать модель для приложения, используя объекты CLR (Common Language Runtime) и специальные объекты POCO (Plain Old CLR Object).

При проектировании приложений с подходом Code-First, сначала создаются классы модели данных, не обращая никакого внимания на Entity Framework. После того, как понадобилось работать с базой данных, используются различные инструменты, которые проецируют структуру базы данных из созданной модели классов. После этого можно вернуться к этой модели в коде и, например, изменить ее. Эти изменения затем можно будет отразить в базе данных используя все те же инструменты [4].

Важным нововведением версии Entity Framework 5 в плане подхода Code-First, является то, что созданная модель классов теперь сразу является сущностной моделью данных EDM (Entity Data Model), поэтому отпала необходимость использовать файл EDMX. В более ранних версиях разработчику, использующему подход Code-First, приходилось добавлять отношения между моделью классов и файлом EDMX, т.е. отображать любые изменения модели сразу в двух местах. Очевидно, что этот подход приводил к появлению кучи ошибок, если разработчик забывал синхронизировать эти изменения в обоих файлах.

Чтобы указать среде Visual Studio, что модель классов является моделью EDM, нужно, во-первых, установить сборки Entity Framework в проект, а во-вторых, добавить класс контекста базы данных, унаследованный от класса DbContext, находящегося в пространстве имен System.Data.Entity.

Не нужно беспокоится о способах взаимодействия с базой данных, Entity Framework определяет ряд вспомогательных методов. Вам не нужно знать деталей подключения и даже имени базы данных [5].

Для того, чтобы получить доступ к базе данных с помощью Entity Framework, нужно использовать поставщики данных, совместимые с Entity Framework. Поставщики данных (data providers) организуют взаимодействие между сущностными классами и системой управления базами данных (СУБД). Фактически они отвечают за то, например, какой код C# нужно сгенерировать из базы данных при использовании подхода Database-First, или какой код SQL нужно создать, при отображении графической модели EDMX на базу данных.

Visual Studio по умолчанию поставляется поставщиком данных SqlClient, что позволяет получить доступ к большинству версий СУБД SQL Server: 2005, 2008, 2010, 2012, 2014. Начиная с версии Visual Studio 2010, Microsoft отказалась от поддержки SQL Server 2000. Можно также создать приложение, обращающееся к SQL Server Compact. Тем не менее, этот провайдер имеет некоторые строгие ограничения, такие как неспособность поддерживать схемы с одинаковыми именами ограничений.

Можно получить доступ к ряду сторонних поставщиков данных для других СУБД. В настоящее время поставщики данных Entity Framework поддерживаются следующими СУБД: MySQL, Oracle, Progress, VistaDB, Devart, OpenLink, SQL Anywhere, Sybase, SQLite, Synergex, Firebird и PostgreSQL (через поставщик Npgsql).

.NET Core – это открытая универсальная платформа разработки, которая поддерживается корпорацией Майкрософт и сообществом .NET на сайте GitHub. Она является кроссплатформенной, поддерживает Windows, Mac OS и Linux и может использоваться на устройствах, в облаке, во внедренных системах и в сценариях IoT (Интернета вещей). В её основе лежат технологии .NET Framework и Silverlight. Она оптимизирована для мобильных и серверных рабочих нагрузок, поскольку обеспечивает поддержку самодостаточных развёртываний XCOPY [6].

При создании .NET одним из главных приоритетов была высокая скорость разработки приложений для Windows. На практике это означает, что разработка .NET была неразрывно связана с Visual Studio. Безусловно, Visual Studio – замечательный инструмент. Он позволяет работать эффективно, а его отладчик – однозначно лучший из тех, которые мне доводилось использовать.

Платформа ASP.NET Core представляет технологию, предназначенную для создания различного рода веб-приложений.

С одной стороны, ASP.NET Core является продолжением развития платформы ASP.NET. Но, с другой стороны, это не просто очередной релиз. Выход ASP.NET Core фактически означает революцию всей платформы, ее качественное изменение.

ASP.NET Core теперь полностью является opensource-фреймворком. Все исходные файлы фреймворка доступны на GitHub.

ASP.NET Core может работать поверх кроссплатформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS, Linux. И таким образом, с помощью ASP.NET Core можно создавать кроссплатформенные приложения. И хотя Windows в качестве среды для разработки и развертывания приложения до сих пор превалирует, но теперь уже нет ограничентй только этой операционной системой. То есть можно запускать веб-приложения не только на ОС Windows, но и на Linux и Mac OS. А для развертывания веб-приложения можно использовать традиционный IIS, либо кроссплатформенный веб-сервер Kestrel [7].

ASP.NET Core включает в себя фреймворк MVC, который объединяет функциональность MVC, Web API и Web Pages. В предыдущих версии платформы данные технологии реализовались отдельно и поэтому содержали много дублирующей функциональности. Сейчас же они объединены в одну программную модель ASP.NET Core MVC. А Web Forms полностью ушли в прошлое. Кроме объединения вышеупомянутых технологий в одну модель в MVC был добавлен ряд дополнительных функций.

Одной из таких функций являются тэг-хелперы (tag helper), которые позволяют более органично соединять синтаксис html с кодом С#.

ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. И можно либо использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.

Также было упрощено управление зависимостями и конфигурирование проекта. Фреймворк теперь имеет свой легковесный контейнер для внедрения зависимостей, и больше нет необходимости применять сторонние контейнеры, такие как Autofac, Ninject. Хотя при желании их также можно продолжать использовать.

В качестве инструментария разработки мы можно использовать последние выпуски Visual Studio, начиная с версии Visual Studio 2015. Кроме того, можно создавать приложения в среде Visual Studio Code, которая является кроссплатформенной и может работать как на Windows, так и на Mac OS X и Linux [8].

Для обработки запросов теперь используется новый конвейер HTTP, который основан на компонентах Katana и спецификации OWIN. А его модульность позволяет легко добавить свои собственные компоненты.

Если суммировать, то можно выделить следующие ключевые отличия ASP.NET Core:

* легковесный и модульный конвейер HTTP-запросов;
* возможность развертывать приложение как на IIS, так и в рамках своего собственного процесса;
* использование платформы .NET Core и ее функциональности;
* распространение пакетов платформы через NuGet;
* интегрированная поддержка для создания и использования пакетов NuGet;
* единый стек веб-разработки, сочетающий Web UI и Web API;
* конфигурация для упрощенного использования в облаке;
* встроенная поддержка для внедрения зависимостей;
* расширяемость;
* кроссплатформенность: возможность разработки и развертывания приложений ASP.NET на Windows, Mac и Linux;
* развитие как open source, открытость к изменениям.

Эти и другие особенности и возможности стали основой для новой модели программирования [9].

Что касается базы данных – здесь применялся язык SQL и система Microsoft SQL Server.

MS SQL Server – это платформа для решения критически важных задач в масштабе предприятия, обладающая высокой доступностью, повышенной производительностью и безопасностью. Решение представляет собой хорошо масштабируемый, полностью реляционный, быстродействующий сервер, способный обрабатывать большие объемы данных для клиент-серверных приложений. Рекордная производительность MS SQL Server обеспечивается новыми технологиями работы с памятью, что помогает предприятиям ускорить свой бизнес и реализовать новые сценарии работы. Кроме того, SQL Server позволяет использовать новые гибридные облачные решения и пользоваться новыми преимуществами облачных вычислений. Расширенные функции безопасности, в сочетании со встроенными, удобными для использования инструментами и управляемым доступом к данным, позволяют организации выполнить требования строгих политик соответствия нормам [10].

1. **ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА**

Главной страницей, на которую попадает пользователь при переходе по ссылке на онлайн-магазин «E-shop», является вкладка сайта «О магазине». На ней находится информация о компании, отражающая показатели работы за последний год, количество сотрудников и торговых площадок, преимущества оформления заказа товаров через данный онлайн-магазин.

В верхней части экрана расположена информация, необходимая пользователю для дальнейшей работы: ссылка на главную вкладку, название онлайн-магазина, контактный номер и ссылки на страницы входа и регистрации.

Ниже размещен перечень категорий товаров, которые пользователь может заказать в данном онлайн-магазине.

На рисунке 3.1 представлена главная страница «О магазине»:

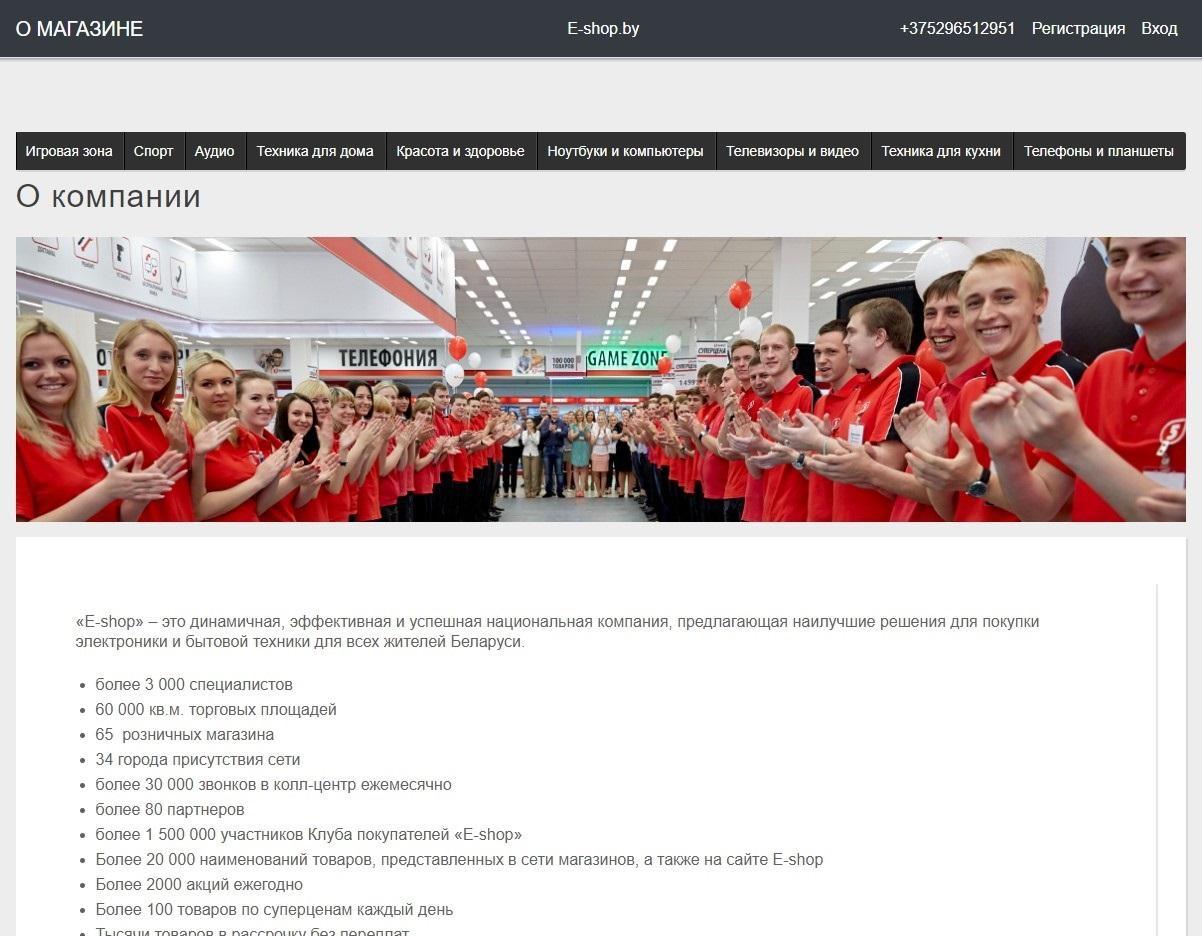


Рисунок 3.1 – Вкладка «О магазине»

При нажатии на название категории пользователь может перейти в любую из них. Для того чтобы просмотреть список товаров и их характеристики, авторизоваться не обязательно.

Пользователь может найти расположение точек самовывоза на карте, либо в списке адресов и времени работы под ней. При наведении на маркер точки самовывоза появляется подсказка с ее адресом. Пример карты изображен на рисунке 3.2:

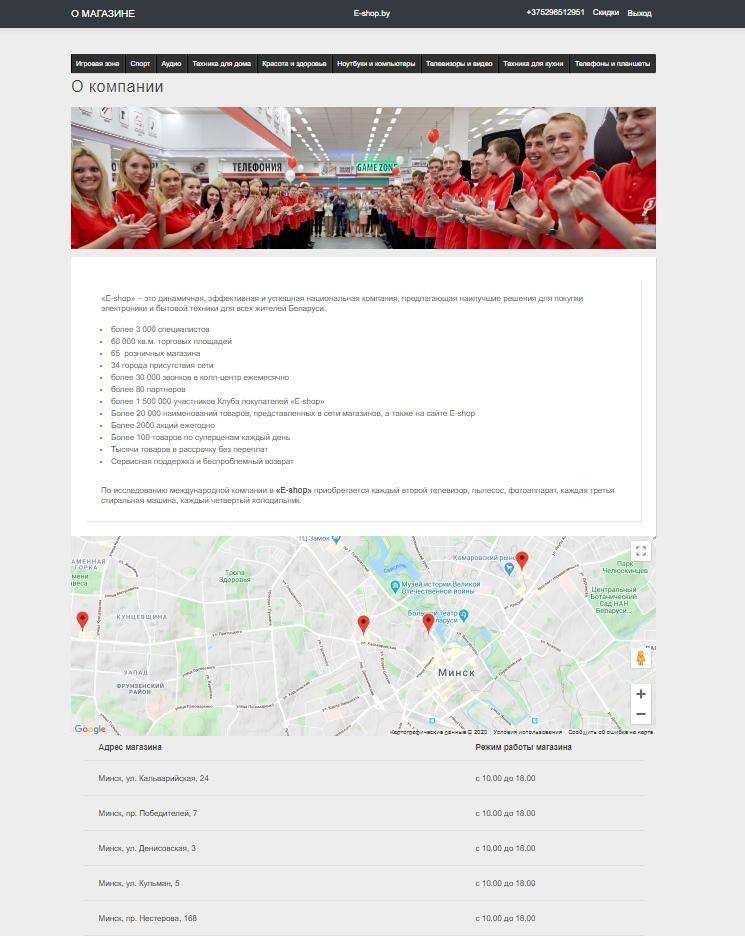


Рисунок 3.2 – Карта точек самовывоза

При переходе в категорию товара пользователь может просмотреть список всех товаров, относящихся к данной категории, отсортировать его, выполнить поиск по названию, отфильтровать список для отображения тех товаров, которые соответствуют критериям запроса покупателя. Выполнить фильтрацию можно по типу товара, производителю, длине, ширине, высоте, цене, цвету, мощности, стране производства, весу и материалу. После нажатия на кнопку «Применить» пользователь увидит обновленный список.

Для каждого товара на экране отображается его название, фото, краткое описание, тип, производитель, мощность и цена.

Пример списка товаров выбранной категории представлен на рисунке 3.3:

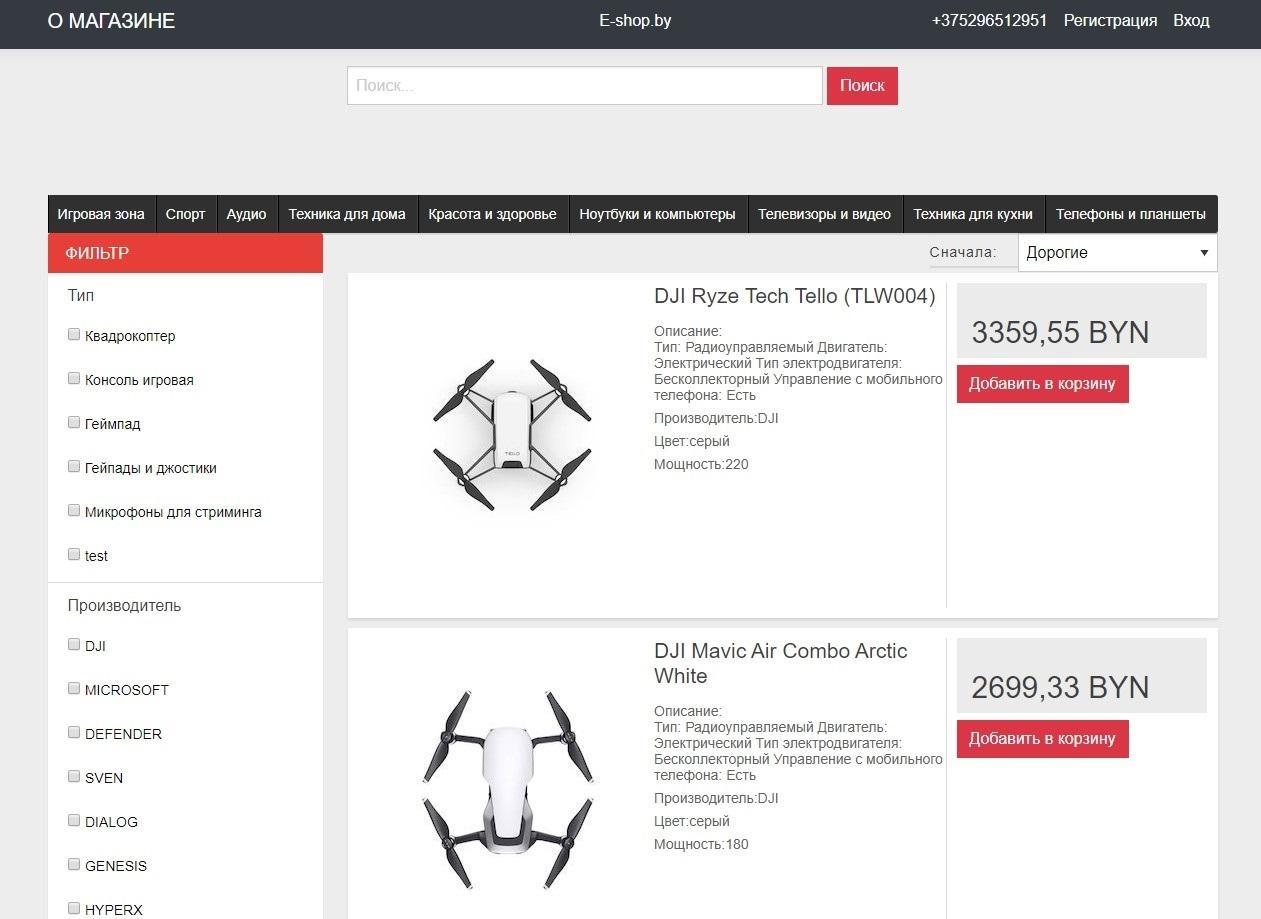


Рисунок 3.3 – Список товаров выбранной категории

При нажатии на товар пользователь может перейти к его более детальному описанию. В открывшейся вкладке отображаются все имеющиеся фотографии товара, а также подробная информация о нем. При помощи кнопки «Вернуться к списку товаров» пользователь может снова перейти на экран категории, в которой он находится.

Пример экрана с детальным описанием товара представлен на рисунке 3.4:

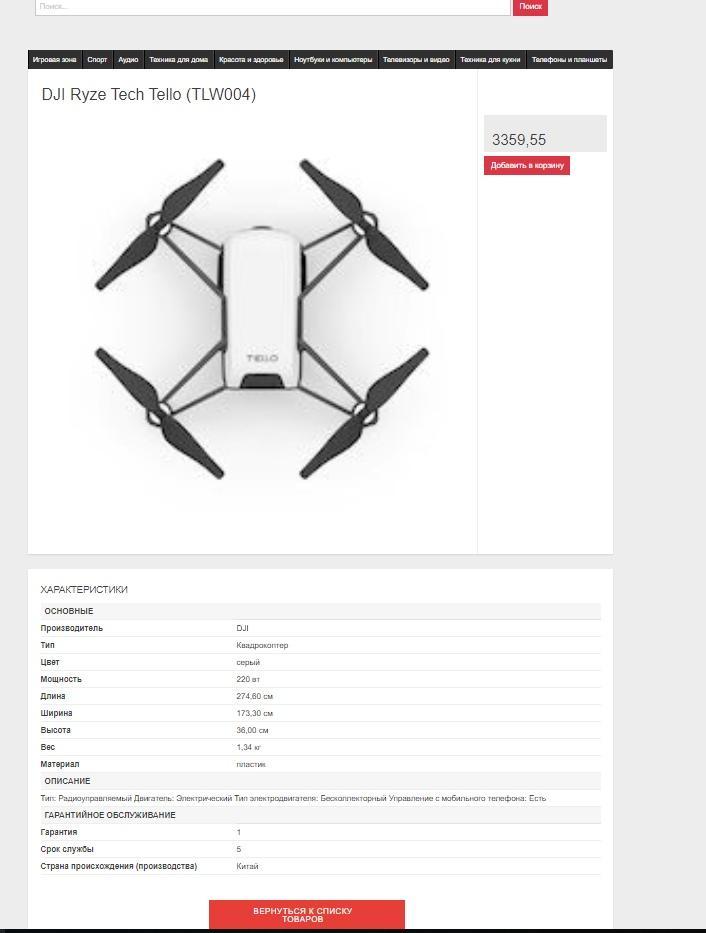


Рисунок 3.4 – Подробное описание товара

Чтобы заказать товар, пользователю необходимо авторизоваться и добавить его в корзину. Он может сделать это, нажав на соответствующую кнопку после входа в личный кабинет. Значение суммы стоимостей товаров, которые пользователь добавил в корзину, увеличится.

Чтобы перейти к оформлению заказа, пользователю необходимо нажать на кнопку «Корзина» в верхней части экрана. В результате он увидит данные товаров, помещенных в корзину, как на рисунке 3.5:

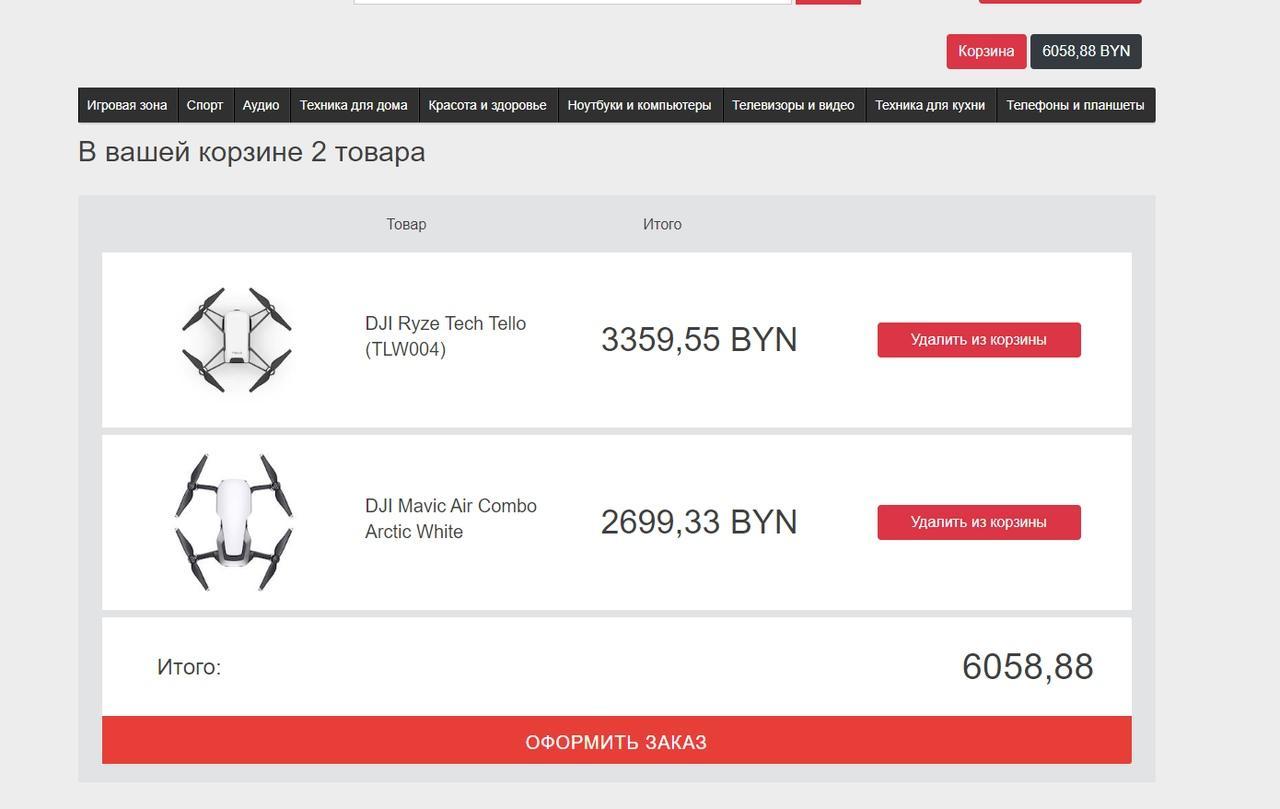


Рисунок 3.5 – Корзина

Пользователь видит количество товаров в корзине, их наименование и стоимость, общую сумму. Реализована возможность удалить товар из корзины, если его покупка уже не актуальна.

При нажатии на кнопку «Оформить заказ» пользователь переходит ко второму этапу, где указывает способ доставки и место получения товара. Адрес точки самовывоза он может выбрать в списке либо на карте (рисунок 3.6), нажав на соответствующий маркер, при наведении на который увидит точный адрес расположения данного магазина:

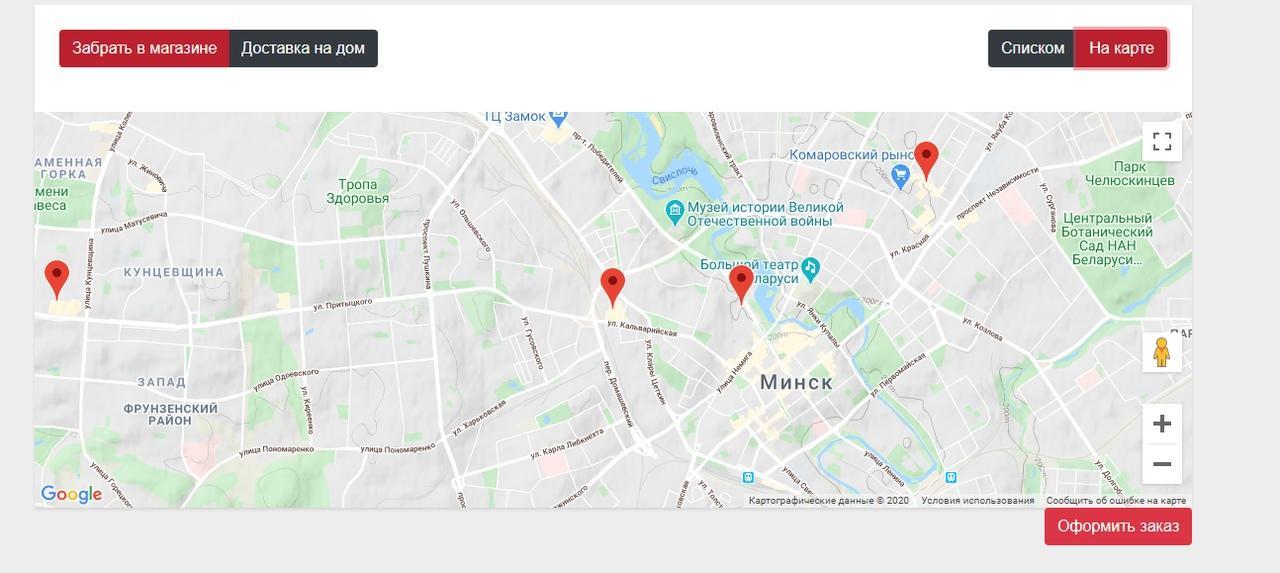


Рисунок 3.6 – Карта точек самовывоза

На этапе выбора адреса заказа пользователь может указать один из сохраненных адресов. Добавить их можно также в личном кабинете. Список адресов пользователя представлен на рисунке 3.7:

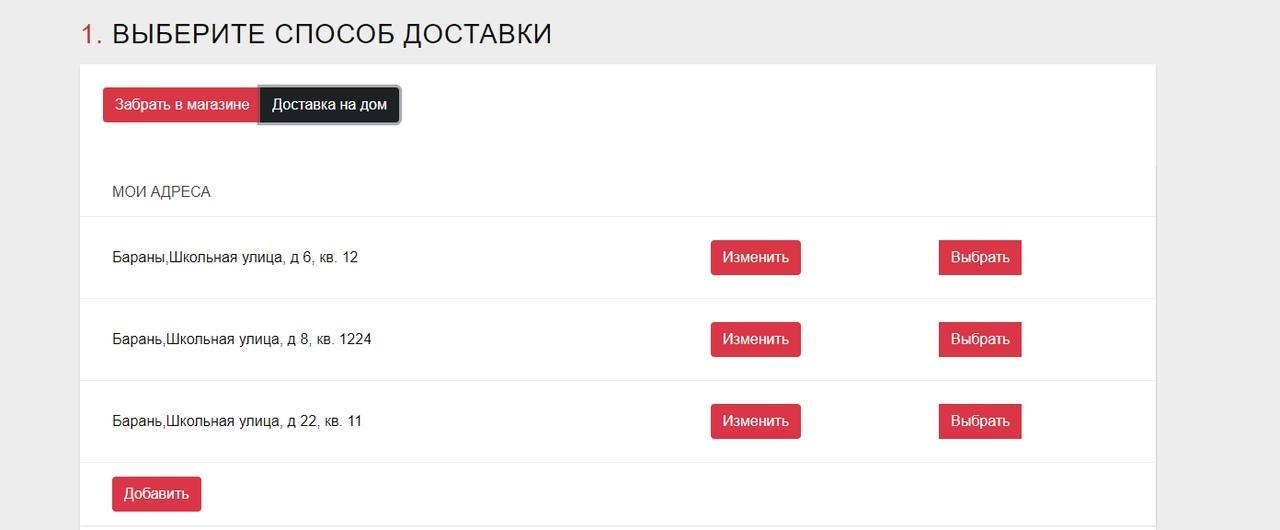


Рисунок 3.7 – Список сохраненных адресов

Если адрес нужно отредактировать, то по нажатию на кнопку «Изменить» пользователь увидит форму, в которой можно ввести другие значения. Такая же форма заполнения адреса появится при его добавлении. Изображение данной формы представлено на рисунке 3.8:

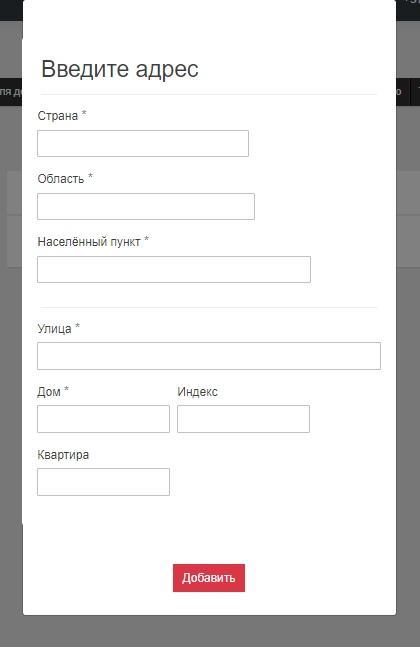


Рисунок 3.8 – Добавление адреса доставки заказа

После выбора адреса доставки заказа пользователь переходит на экран подтверждения. Он может ввести контактные данные, добавить комментарий к заказу, проверить адрес доставки, выбрать дату получения товара и просмотреть детали.

Для того чтобы оформить заказ, пользователю необходимо нажать на соответствующую кнопку, после чего заказ будет зарегистрирован в системе. Пользователь сможет в любой момент увидеть детали заказа в разделе «Мои заказы».

Экран подтверждения оформления заказа изображен на рисунке 3.9:

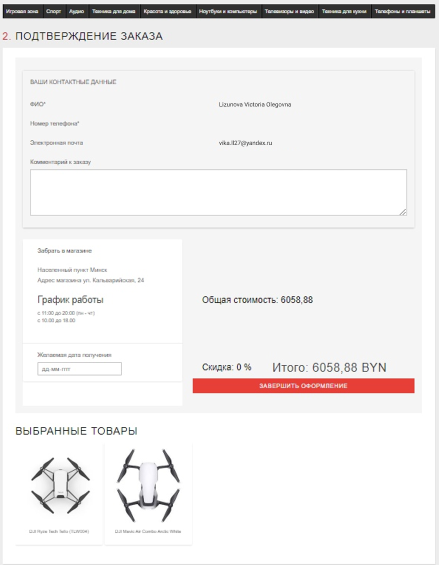


Рисунок 3.9 – Подтверждение заказа

Личный кабинет дает пользователю возможность отслеживать заказы, сохранять адреса доставки, изменять личные данные и иметь скидку, получаемую в зависимости от общей суммы всех заказов. Для того чтобы зарегистрироваться в системе, необходимо перейти на вкладку «Регистрация». Она показана на рисунке 3.10:

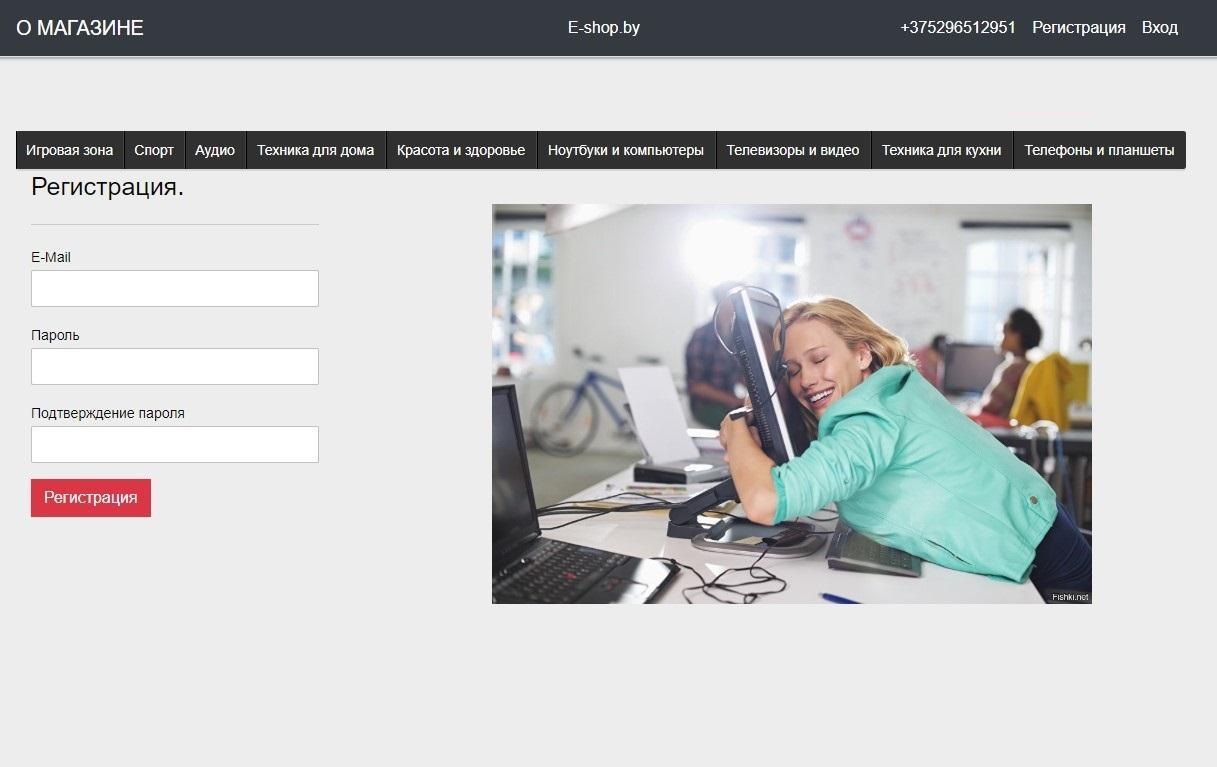


Рисунок 3.10 – Регистрация

Для регистрации пользователю необходимо ввести личную почту, придумать пароль и подтвердить его. Система проверит, не зарегистрирован ли такой пользователь, и, если валидация данных прошла успешно, пользователь сможет войти в личный аккаунт. Если пользователь ввел слабый пароль, система потребует генерации более сложного.

В профиле пользователя ему предоставляются такие возможности:

* редактирование личных данных;
* смена пароля;
* управление заказами;
* управление адресами доставки.

Значение своей скидки пользователь также может найти в личном кабинете.

Для администратора онлайн-магазина предоставляемый функционал шире: он также может добавлять и удалять категории товаров, изменять данные существующих товаров и добавлять новые, удалять их. Такие возможности реализуемы при помощи соответствующих кнопок в разделах категорий товаров.

Именно администратор может управлять программой лояльности: задавать начальные и конечные значения общей суммы заказов, в соответствии с которой пользователь может получить ту или иную скидку. Эта возможность реализована на вкладке «Скидки», представленной на рисунке 3.11:

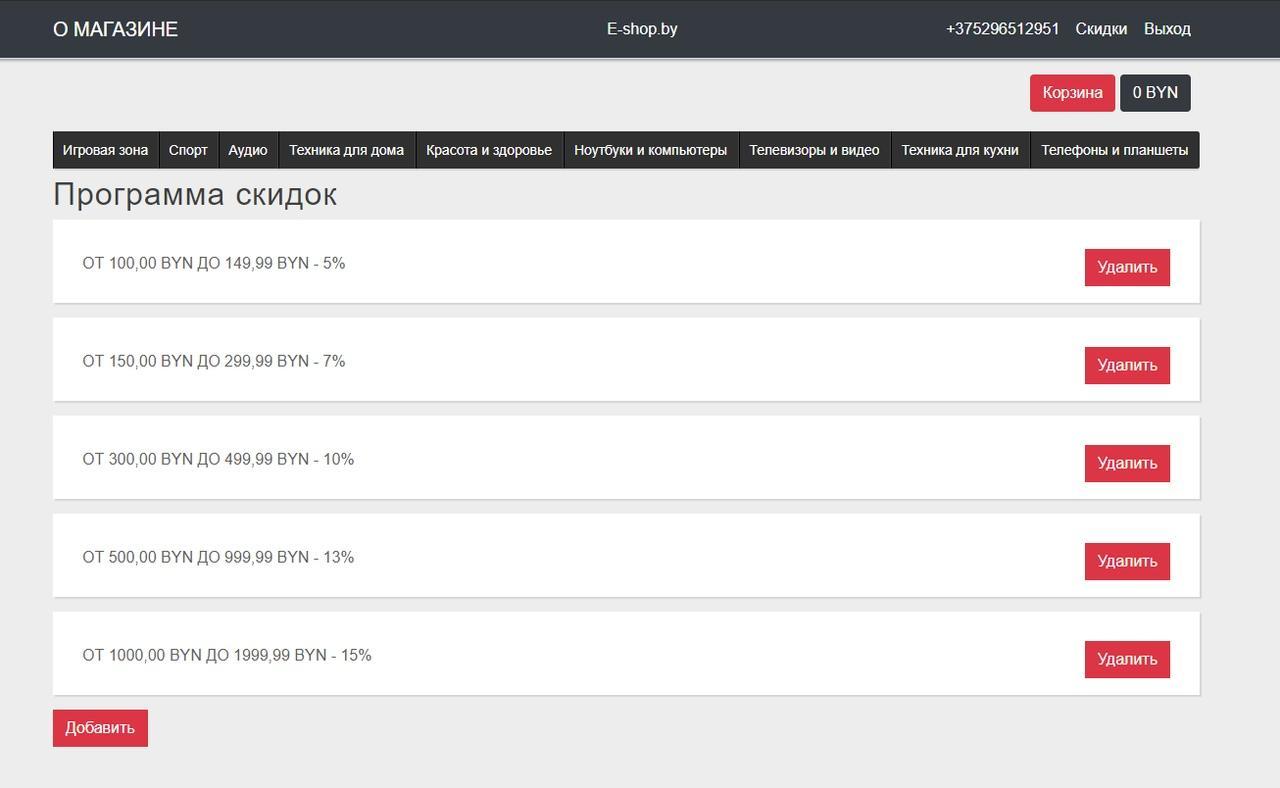


Рисунок 3.11 – Управление программой лояльности

Рассмотрев реализацию вариантов использования системы пользователем и администратором, можно сделать вывод, что все требования, предъявленные к функционалу системы, были полностью реализованы. Система предоставляет все возможности, необходимые для успешной работы онлайн-магазина, а также ограничивает пользователя от совершения ошибок, которые могут возникнуть во время работы над заказом товаров.

1. **РУКОВОДСТВО ПО РАЗВЕРТЫВАНИЮ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА**

В рамках данного раздела представлен алгоритм по развёртыванию системы на стороннем сервере.

Состоит данный раздел из трёх пунктов:

* требования к системе;
* развёртывание БД на сервере СУБД;
* развёртывание приложения на сервере.

Что касается требований к системе, то их можно представить в виде следующего списка:

* поддержка языка C#;
* поддержка СУБД MSSQL;
* поддержка формата XHTML для отображения.

Естественно, все предъявляемые требования характеризуют поддержку версий технологий, используемых при создании приложения. Данные требования выполняются на большинстве веб-серверов.

Что касается второго пункта – это развёртывание БД на сервере. Его можно описать следующим алгоритмом:

1 Запустить интерфейс управления выбранной СУБД на сервере.

2 Создать схему Lizunova\_V\_O, если она не создана. Выбрать её.

3 Выбрать в меню пункт импорта данных и импортировать файл Lizunova\_v\_o.sql из корневого каталога проекта.

Создание базы данных завершено.

Алгоритм развёртывания приложения на сервере представлен далее:

1. Первым делом следует опубликовать проект при помощи кнопки «Publish» в меню проекта в Microsoft Visual Studio 2019.
2. Вторым действием необходимо выполнить настройку публикации, выбрать ее цель и заполнить остальные данные.
3. Выполнить сборку проекта.
4. Далее необходима настройка соединения с БД, которая, предполагается, уже развёрнута в рамках второго пункта данного раздела.
5. Посредством браузера запустить в адресной строке URL-адрес домена и протестировать работу сайта.

После этого развёртывание приложения завершено. Система полностью готова и функционирует.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения данного курсового проекта был спроектирован и разработан онлайн-магазин. Данный проект позволяет совершать покупки, находясь в любом месте, где есть доступ в интернет и устройство для запуска сайта.

В онлайн-магазине авторизация и регистрация является дополнительной функцией. Пользователь может просматривать товары, не прибегая к регистрации. Личный кабинет является вспомогательным инструментом пользователя. Благодаря личному кабинету пользователь может создавать заказ, отслеживать историю покупок, заранее ввести личные данные и адреса доставки, следить за изменением личной скидки.

Администратор сайта имеет полные права на управления онлайн-магазином, начиная от редактирования категорий и данных товаров, заканчивая управлением программой лояльности.

Скрипт базы данных выполнен на языке SQL в СУБД MSSQL, основным языком серверной части приложения является C#, клиентская часть приложения реализована с помощью JavaScript, HTML и CSS.

В процессе выполнения курсового проекта были спроектированы функциональные модели системы «как-есть» и «как-должно-быть» в нотации IDEF0. Для проектирования функциональности системы были созданы UML диаграммы последовательности, компонентов, развертывания и состояния. Также спроектированы блок-схемы алгоритмов работы системы. Для создания базы данных была построена ее модель.

Исходя из оценки проделанной работы, можно сделать вывод о том, что поставленные в начале проектирования цель и задачи были успешно достигнуты и выполнены. В ходе выполнения данной курсовой работы были изучены основные аспекты и возможности языка C#, базы данных MSSQL, а также освоены на практике возможности технологии Entity Framework и платформы .NET Core.

Для пользователя создан максимально простой, интуитивно понятный и удобный интерфейс. Доступ к личным данным осуществляется посредством авторизации пользователя. Приложение обеспечивает высокий контроль учета и хранения данных. Разработанная система имеет практическое применение, т.к. онлайн-заказ товаров в современных реалиях как никогда актуален.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Инфопедия [Электронный ресурс]. – Режим доступа: https://infopedia.su;

[2] Фролов, Основы работы с AllFusion Process Modeler: методические указания по дисциплине «Автоматизированное проектирование информационных систем» для студентов направления 230100 «Информатика и вычислительная техника» / сост. В. А. Фролов. – Ульяновск: УлГТУ, 2014 – 39 с.;

[3] Изучение SQL с помощью MSSQL [Электронный ресурс]. – Режим доступа: https://mssql-tutorial.ru;

[4] Грейди Буч, Джеймс Рамбо, Айвар Джекобсон. Язык UML. Руководство пользователя – 2-е изд. – М., СПб.: ДМК Пресс, Питер, 2004. – 432 с.;

[5] Бейтс, Б. Head first. Паттерны проектирования/ Б. Бейтс – Санкт-Перербург: Питер, 2016. – 540 с.;

[6] Введение в ASP.NET Core [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/aspnet5/1.1.php.

[7] Ускоряемся в Entity Framework Core [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/487734/

[8] База данных [Электронный ресурс]. – Электронные данные. – Режим доступа: https://hostiq.ua/wiki/database/;

[9] Цели применения диаграммы вариантов использования [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=4

[10] Назначение диаграммы вариантов использования [Электронный ресурс]. – Электронные данные. – Режим доступа: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema12/tema12\_2

[11] Информационная система» [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Информационная\_система.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Отчет о проверке на заимствования в системе «Антиплагиат»**

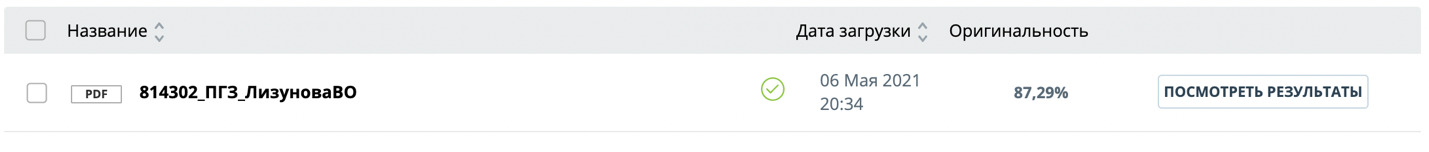


Рисунок А.1 – Отчет о проверке на заимствования в системе «Антиплагиат»

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Листинг кода алгоритмов, реализующих бизнес-логику**

AppDomainContextBase.cs:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using BSUIR.DAL.Interfaces;

using BSUIR.DAL.Interfaces.Models;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

namespace BSUIR.DAL.Context

{

public abstract class AppDomainContextBase<TDbContext>

where TDbContext : IdentityDbContext<User>

{

private readonly TDbContext \_dbContext;

public AppDomainContextBase(TDbContext dbContext)

{

\_dbContext = dbContext ?? throw new ArgumentNullException(nameof(dbContext));

}

public async Task<int> SaveChangesAsync() => await \_dbContext.SaveChangesAsync();

public async Task<int> SaveChangesAsync(CancellationToken cancellationToken) =>

await \_dbContext.SaveChangesAsync(cancellationToken);

protected IEntitySet<TEntityType> GetEntitySet<TEntityType>()

where TEntityType : class

{

var entitySetConstructedType = typeof(EntitySet<>).MakeGenericType(typeof(TEntityType));

var dbSetGenericPropertyValue = TypeDescriptor

.GetProperties(\_dbContext)

.Cast<PropertyDescriptor>()

.Where(p =>

p.PropertyType.IsConstructedGenericType && p.PropertyType.GetGenericTypeDefinition() ==

typeof(DbSet<>)

&& p.PropertyType.GenericTypeArguments[0] ==

typeof(TEntityType)).FirstOrDefault().GetValue(\_dbContext);

var entitySet = Activator.CreateInstance(entitySetConstructedType, dbSetGenericPropertyValue);

Продолжение приложения Б

return entitySet as IEntitySet<TEntityType>;

}

}

}

QuaryableExtensions.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using System.Threading.Tasks;

using CoreExtensions = Microsoft.EntityFrameworkCore.EntityFrameworkQueryableExtensions;

namespace BSUIR.DAL.Interfaces.Extensions.Quaryable

{

public static class QuaryableExtensions

{

public static Task<List<T>> ToListAsync<T>(this IQueryable<T> source)

where T : class

{

if (source is EntitySet<T>)

{

source = (source as EntitySet<T>).DbSet;

}

return CoreExtensions.ToListAsync(source);

}

public static Task<T[]> ToArrayAsync<T>(this IQueryable<T> source)

where T : class

{

if (source is EntitySet<T>)

{

source = (source as EntitySet<T>).DbSet;

}

return CoreExtensions.ToArrayAsync(source);

}

public static IQueryable<TEntity> Include<TEntity, TRelatedEntityProperty>(this IQueryable<TEntity> source, Expression<Func<TEntity, TRelatedEntityProperty>> expression)

where TEntity : class

where TRelatedEntityProperty : class

{

if(source is EntitySet<TEntity>)

{

source = (source as EntitySet<TEntity>).DbSet;

}

return CoreExtensions.Include<TEntity, TRelatedEntityProperty>(source, expression);

}

Продолжение приложения Б

public static IQueryable<TEntity> Include<TEntity>(this IQueryable<TEntity> source, string navigationPropertyPath)

where TEntity : class

{

if (source is EntitySet<TEntity>)

{

source = (source as EntitySet<TEntity>).DbSet;

}

return CoreExtensions.Include<TEntity>(source, navigationPropertyPath);

}

}

}

CustomerService.cs:

using System;

using System.Linq;

using System.Threading.Tasks;

using AutoMapper;

using BSUIR.BL.Interfaces.Models.Customers;

using BSUIR.BL.Interfaces.Services;

using BSUIR.DAL.Interfaces.Context;

using BSUIR.DAL.Interfaces.Extensions.Quaryable;

using CustomerDTO = BSUIR.DAL.Interfaces.Models.Customer;

namespace BSUIR.BL.Services

{

public class CustomerService : ICustomerService

{

private readonly ICustomersContext \_customersContext;

private readonly IMapper \_mapper;

public CustomerService(IMapper mapper, ICustomersContext customersContext)

{

\_mapper = mapper;

\_customersContext = customersContext;

}

public async Task<TMapTo> CreateCustomerAsync<TMapTo>(Customer customerCreate) where TMapTo : Customer

{

try

{ var mappedDtoModel = \_mapper.Map<Customer, CustomerDTO>(customerCreate);

var createdEntity = await \_customersContext.Customers.AddAsync(mappedDtoModel);

if (createdEntity is null)

{

throw new Exception(

$"Can not create entity {typeof(CustomerDTO)} from parameter {nameof(customerCreate)}");

}

await \_customersContext.SaveChangesAsync();

var mappedCreatedEntity = \_mapper.Map<CustomerDTO, TMapTo>(createdEntity);

Продолжение приложения Б

return mappedCreatedEntity;

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

public async Task<TMapTo> UpdateCustomerAsync<TMapTo>(Customer customerToUpdate) where TMapTo : Customer

{

try

{

var address = await this.GetCustomerByIdAsync(customerToUpdate.Id);

var mappedDtoModel = \_mapper.Map<Customer, CustomerDTO>(customerToUpdate, address);

var updatedModel = \_customersContext.Customers.Update(mappedDtoModel);

if (updatedModel is null)

{

throw new Exception(

$"Can not update entity {typeof(CustomerDTO)} using data from object parameter {nameof(customerToUpdate)}");

}

await \_customersContext.SaveChangesAsync();

return \_mapper.Map<TMapTo>(updatedModel);

}

catch (Exception ex)

{

throw new Exception(

"Some error occured while updating customer. Updated changes won't be applied", ex);

}

}

public async Task<TMapTo> GetCustomerByIdAsync<TMapTo>(string id) where TMapTo : Customer

{

var foundCustomer = await this.GetCustomerByIdAsync(id);

return \_mapper.Map<TMapTo>(foundCustomer);

}

private async Task<CustomerDTO> GetCustomerByIdAsync(string customerId)

{

if (!\_customersContext.Customers.Any(p => p.Id == customerId))

{

throw new Exception($"Customer with id: {customerId} not found.");

}

var addressWithCorrespondedId =

await \_customersContext.Customers.Where(p => p.Id == customerId).ToArrayAsync();

if (addressWithCorrespondedId.Length > 1)

{

throw new Exception(

$"More than one entity of type {typeof(CustomerDTO)} with id {customerId} where found");

}

Продолжение приложения Б

var foundAddress = addressWithCorrespondedId[0];

return foundAddress;

}

}

}

**ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Листинг скрипта генерации базы данных**

USE [Lizunova\_V\_O]

GO

/\*\*\*\*\*\* Object: Table [dbo].[\_\_EFMigrationsHistory] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[\_\_EFMigrationsHistory](

[MigrationId] [nvarchar](150) NOT NULL,

[ProductVersion] [nvarchar](32) NOT NULL,

CONSTRAINT [PK\_\_\_EFMigrationsHistory] PRIMARY KEY CLUSTERED

(

[MigrationId] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[address] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[address](

[id] [int] IDENTITY(1,1) NOT NULL,

[country] [nvarchar](50) NOT NULL,

[city] [nvarchar](50) NOT NULL,

[street] [nvarchar](50) NOT NULL,

[house] [int] NOT NULL,

[post\_index] [int] NOT NULL,

[region] [nvarchar](50) NOT NULL,

[flat] [int] NULL,

[customer\_id] [nvarchar](450) NOT NULL,

CONSTRAINT [PK\_address] PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[AspNetRoleClaims] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[AspNetRoleClaims](

[Id] [int] IDENTITY(1,1) NOT NULL,

[RoleId] [nvarchar](450) NOT NULL,

[ClaimType] [nvarchar](max) NULL,

Продолжение приложения В

[ClaimValue] [nvarchar](max) NULL,

CONSTRAINT [PK\_AspNetRoleClaims] PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[AspNetRoles] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[AspNetRoles](

[Id] [nvarchar](450) NOT NULL,

[Name] [nvarchar](256) NULL,

[NormalizedName] [nvarchar](256) NULL,

[ConcurrencyStamp] [nvarchar](max) NULL,

CONSTRAINT [PK\_AspNetRoles] PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[AspNetUserClaims] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[AspNetUserClaims](

[Id] [int] IDENTITY(1,1) NOT NULL,

[UserId] [nvarchar](450) NOT NULL,

[ClaimType] [nvarchar](max) NULL,

[ClaimValue] [nvarchar](max) NULL,

CONSTRAINT [PK\_AspNetUserClaims] PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[AspNetUserLogins] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[AspNetUserLogins](

[LoginProvider] [nvarchar](450) NOT NULL,

[ProviderKey] [nvarchar](450) NOT NULL,

[ProviderDisplayName] [nvarchar](max) NULL,

[UserId] [nvarchar](450) NOT NULL,

CONSTRAINT [PK\_AspNetUserLogins] PRIMARY KEY CLUSTERED

(

[LoginProvider] ASC,

[ProviderKey] ASC

Продолжение приложения В

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[AspNetUserRoles] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[AspNetUserRoles](

[UserId] [nvarchar](450) NOT NULL,

[RoleId] [nvarchar](450) NOT NULL,

CONSTRAINT [PK\_AspNetUserRoles] PRIMARY KEY CLUSTERED

(

[UserId] ASC,

[RoleId] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[AspNetUsers] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[AspNetUsers](

[Id] [nvarchar](450) NOT NULL,

[UserName] [nvarchar](256) NULL,

[NormalizedUserName] [nvarchar](256) NULL,

[Email] [nvarchar](256) NULL,

[NormalizedEmail] [nvarchar](256) NULL,

[EmailConfirmed] [bit] NOT NULL,

[PasswordHash] [nvarchar](max) NULL,

[SecurityStamp] [nvarchar](max) NULL,

[ConcurrencyStamp] [nvarchar](max) NULL,

[PhoneNumber] [nvarchar](max) NULL,

[PhoneNumberConfirmed] [bit] NOT NULL,

[TwoFactorEnabled] [bit] NOT NULL,

[LockoutEnd] [datetimeoffset](7) NULL,

[LockoutEnabled] [bit] NOT NULL,

[AccessFailedCount] [int] NOT NULL,

CONSTRAINT [PK\_AspNetUsers] PRIMARY KEY CLUSTERED

(

[Id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[AspNetUserTokens] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[AspNetUserTokens](

[UserId] [nvarchar](450) NOT NULL,

[LoginProvider] [nvarchar](450) NOT NULL,

Продолжение приложения В

[Name] [nvarchar](450) NOT NULL,

[Value] [nvarchar](max) NULL,

CONSTRAINT [PK\_AspNetUserTokens] PRIMARY KEY CLUSTERED

(

[UserId] ASC,

[LoginProvider] ASC,

[Name] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[category] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[category](

[id] [int] IDENTITY(1,1) NOT NULL,

[name] [nvarchar](50) NOT NULL,

CONSTRAINT [PK\_category] PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[customer] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[customer](

[id] [nvarchar](450) NOT NULL,

[firstname] [nvarchar](50) NULL,

[lastname] [nvarchar](50) NULL,

[secondname] [nvarchar](50) NULL,

[birthdate] [date] NULL,

[mobile\_number] [nvarchar](50) NULL,

[sex] [nvarchar](50) NULL,

[discount] [int] NULL,

CONSTRAINT [PK\_customer] PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[delivery\_address] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[delivery\_address](

[id] [int] IDENTITY(1,1) NOT NULL,

[country] [nvarchar](50) NOT NULL,

[region] [nvarchar](50) NOT NULL,

Продолжение приложения В

[city] [nvarchar](50) NOT NULL,

[street] [nvarchar](50) NOT NULL,

[house] [int] NOT NULL,

[flat] [int] NULL,

[post\_index] [int] NOT NULL,

[working\_hours] [nvarchar](100) NOT NULL,

[coordinates] [nvarchar](200) NOT NULL,

CONSTRAINT [PK\_delivery\_address] PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[discounts] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[discounts](

[id] [int] IDENTITY(1,1) NOT NULL,

[from] [decimal](10, 2) NOT NULL,

[to] [decimal](10, 2) NOT NULL,

[amount] [int] NULL,

CONSTRAINT [PK\_discounts] PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[item] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[item](

[id] [int] IDENTITY(1,1) NOT NULL,

[name] [nvarchar](50) NOT NULL,

[producer] [nvarchar](50) NULL,

[power] [int] NOT NULL,

[length] [decimal](12, 2) NOT NULL,

[height] [decimal](12, 2) NOT NULL,

[width] [decimal](12, 2) NOT NULL,

[color] [nvarchar](50) NOT NULL,

[guarantee] [int] NULL,

[validity] [int] NULL,

[country] [nvarchar](50) NOT NULL,

[material] [nvarchar](50) NULL,

[description] [nvarchar](4000) NULL,

[weight] [decimal](14, 2) NULL,

[type] [nvarchar](50) NOT NULL,

[date] [datetime] NOT NULL,

[price] [decimal](12, 2) NOT NULL,

[category\_id] [int] NOT NULL,

CONSTRAINT [PK\_item] PRIMARY KEY CLUSTERED

(

Продолжение приложения В

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[order] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[order](

[id] [int] IDENTITY(1,1) NOT NULL,

[date] [date] NOT NULL,

[status] [nvarchar](50) NOT NULL,

[comment] [nvarchar](500) NULL,

[amount] [decimal](12, 2) NOT NULL,

[customer\_id] [nvarchar](450) NOT NULL,

[delivery\_address\_id] [int] NULL,

[address\_id] [int] NULL,

CONSTRAINT [PK\_order] PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[order\_has\_item] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[order\_has\_item](

[order\_id] [int] NOT NULL,

[item\_id] [int] NOT NULL,

[id] [int] IDENTITY(1,1) NOT NULL,

CONSTRAINT [PK\_order\_has\_item] PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[photo] Script Date: 14.05.2020 10:07:08 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[photo](

[id] [int] IDENTITY(1,1) NOT NULL,

[name] [nvarchar](50) NULL,

[item\_id] [int] NOT NULL,

[link] [nvarchar](200) NOT NULL,

CONSTRAINT [PK\_photo] PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

Продолжение приложения В

) ON [PRIMARY]

GO

ALTER TABLE [dbo].[address] WITH CHECK ADD CONSTRAINT [FK\_address\_customer] FOREIGN KEY([customer\_id])

REFERENCES [dbo].[customer] ([id])

GO

ALTER TABLE [dbo].[address] CHECK CONSTRAINT [FK\_address\_customer]

GO

ALTER TABLE [dbo].[AspNetRoleClaims] WITH CHECK ADD CONSTRAINT [FK\_AspNetRoleClaims\_AspNetRoles\_RoleId] FOREIGN KEY([RoleId])

REFERENCES [dbo].[AspNetRoles] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[AspNetRoleClaims] CHECK CONSTRAINT [FK\_AspNetRoleClaims\_AspNetRoles\_RoleId]

GO

ALTER TABLE [dbo].[AspNetUserClaims] WITH CHECK ADD CONSTRAINT [FK\_AspNetUserClaims\_AspNetUsers\_UserId] FOREIGN KEY([UserId])

REFERENCES [dbo].[AspNetUsers] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[AspNetUserClaims] CHECK CONSTRAINT [FK\_AspNetUserClaims\_AspNetUsers\_UserId]

GO

ALTER TABLE [dbo].[AspNetUserLogins] WITH CHECK ADD CONSTRAINT [FK\_AspNetUserLogins\_AspNetUsers\_UserId] FOREIGN KEY([UserId])

REFERENCES [dbo].[AspNetUsers] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[AspNetUserLogins] CHECK CONSTRAINT [FK\_AspNetUserLogins\_AspNetUsers\_UserId]

GO

ALTER TABLE [dbo].[AspNetUserRoles] WITH CHECK ADD CONSTRAINT [FK\_AspNetUserRoles\_AspNetRoles\_RoleId] FOREIGN KEY([RoleId])

REFERENCES [dbo].[AspNetRoles] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[AspNetUserRoles] CHECK CONSTRAINT [FK\_AspNetUserRoles\_AspNetRoles\_RoleId]

GO

ALTER TABLE [dbo].[AspNetUserRoles] WITH CHECK ADD CONSTRAINT [FK\_AspNetUserRoles\_AspNetUsers\_UserId] FOREIGN KEY([UserId])

REFERENCES [dbo].[AspNetUsers] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[AspNetUserRoles] CHECK CONSTRAINT [FK\_AspNetUserRoles\_AspNetUsers\_UserId]

GO

ALTER TABLE [dbo].[AspNetUserTokens] WITH CHECK ADD CONSTRAINT [FK\_AspNetUserTokens\_AspNetUsers\_UserId] FOREIGN KEY([UserId])

REFERENCES [dbo].[AspNetUsers] ([Id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[AspNetUserTokens] CHECK CONSTRAINT [FK\_AspNetUserTokens\_AspNetUsers\_UserId]

GO

ALTER TABLE [dbo].[customer] WITH CHECK ADD CONSTRAINT [FK\_customer\_user] FOREIGN KEY([id])

REFERENCES [dbo].[AspNetUsers] ([Id])

GO

ALTER TABLE [dbo].[customer] CHECK CONSTRAINT [FK\_customer\_user]

GO

ALTER TABLE [dbo].[item] WITH CHECK ADD CONSTRAINT [FK\_item\_category] FOREIGN KEY([category\_id])

Продолжение приложения В

REFERENCES [dbo].[category] ([id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[item] CHECK CONSTRAINT [FK\_item\_category]

GO

ALTER TABLE [dbo].[item] WITH CHECK ADD CONSTRAINT [FK\_item\_item] FOREIGN KEY([id])

REFERENCES [dbo].[item] ([id])

GO

ALTER TABLE [dbo].[item] CHECK CONSTRAINT [FK\_item\_item]

GO

ALTER TABLE [dbo].[order] WITH CHECK ADD CONSTRAINT [FK\_order\_address] FOREIGN KEY([address\_id])

REFERENCES [dbo].[address] ([id])

GO

ALTER TABLE [dbo].[order] CHECK CONSTRAINT [FK\_order\_address]

GO

ALTER TABLE [dbo].[order] WITH CHECK ADD CONSTRAINT [FK\_order\_customer] FOREIGN KEY([customer\_id])

REFERENCES [dbo].[customer] ([id])

GO

ALTER TABLE [dbo].[order] CHECK CONSTRAINT [FK\_order\_customer]

GO

ALTER TABLE [dbo].[order] WITH CHECK ADD CONSTRAINT [FK\_order\_delivery\_address\_id] FOREIGN KEY([delivery\_address\_id])

REFERENCES [dbo].[delivery\_address] ([id])

GO

ALTER TABLE [dbo].[order] CHECK CONSTRAINT [FK\_order\_delivery\_address\_id]

GO

ALTER TABLE [dbo].[order\_has\_item] WITH CHECK ADD CONSTRAINT [FK\_order\_has\_item\_item] FOREIGN KEY([item\_id])

REFERENCES [dbo].[item] ([id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[order\_has\_item] CHECK CONSTRAINT [FK\_order\_has\_item\_item]

GO

ALTER TABLE [dbo].[order\_has\_item] WITH CHECK ADD CONSTRAINT [FK\_order\_has\_item\_order] FOREIGN KEY([order\_id])

REFERENCES [dbo].[order] ([id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[order\_has\_item] CHECK CONSTRAINT [FK\_order\_has\_item\_order]

GO

ALTER TABLE [dbo].[photo] WITH CHECK ADD CONSTRAINT [FK\_photo\_item] FOREIGN KEY([item\_id])

REFERENCES [dbo].[item] ([id])

ON DELETE CASCADE

GO

ALTER TABLE [dbo].[photo] CHECK CONSTRAINT [FK\_photo\_item]

GO