

РЕФЕРАТ

ПРОГРАММНОЕ СРЕДСТВО: «ИНТЕРНЕТ-МАГАЗИН ОАО «СВІТА-НАК» НА ПЛАТФОРМЕ NODE.JS: дипломный проект / А.П. Авдей. – БГУИР, 2021, – п.з. – 123., чертежей (плакатов) – 6 л. Формата А1.

Объектом исследования данного дипломного проекта является автоматизация процессов покупки товаров определенной организации.

Цель работы – проектирование и разработка программного средства, решающего задачи автоматизации процессов покупки товара, демонстрации пользователю каталога продукции, реализации возможности совершения заказа продукции, сбора и анализа результатов посещения интернет-магазина.

Проведен анализ предметной области и сравнительный анализ аналогов программного средства. На основании проведенного анализа выполнена постановка задачи для дипломного проектирования.

Выполнено моделирование предметной области, в ходе которого разработаны функциональная модель системы в виде диаграммы вариантов использования и информационная модель.

Выполнено проектирование программного средства, в ходе которого разработана проектная документация. На основании проектной документации выполнена разработка программного средства средствами языков программирования JavaScript и Python. Для хранения данных использована СУБД PostgreSQL. Разработана методика использования программного средства.

Всего в пояснительной записке приведено 7 разделов.

Раздел «Аналитический обзор программных продуктов, методов и подходов по теме дипломного проектирования» описывает поставленную задачу для реализации программного средства, цели и задачи дипломного проекта.

Раздел «Моделирование предметной области, разработка функциональных требований и составление их спецификации» описывает процесс моделирования предметной области, функциональную, инфологическую модель, а также спецификацию функциональных и технических требований.

Раздел «Проектирование архитектуры программного средства» описывает архитектуру программного средства, логическую структуру, а также описание некоторых алгоритмов, используемых программным средством.

Раздел «Разработка программного средства»

Раздел «Тестирование программного средства» описывает основные ошибки, которые возникали на этапе разработки и методы их устранения.

Раздел «Руководство пользователя» содержит описание процесса развертывания сервера приложения, среды применения программы и руководство пользователя.

Раздел «Технико-экономическое обоснование разработки интернет-магазина для ОАО «Світанак»» содержит расчеты затрат, связанных с разработкой проекта, а также рентабельности разработки проекта. Проведенные расчеты показали экономическую целесообразность проекта.

СОДЕРЖАНИЕ

Введение.....	8
1 Аналитический обзор программных продуктов, методов и подходов по теме дипломного проектирования.....	9
1.1 Основные понятия и определения в области электронной коммерции ..	9
1.2 Обзор тенденций в области Интернет-торговли.....	11
1.3 Анализ существующих программных решений по теме дипломного проектирования	13
1.4 Постановка целей и задач дипломного проектирования	24
2 Моделирование предметной области, разработка функциональных требований и составление их спецификации	25
2.1 Общие сведения и требования к работе программного продукта	25
2.2 Описание функциональности программного продукта	25
2.3 Разработка информационной модели	30
2.4 Разработка модели взаимодействия пользователя с интерфейсом	32
2.5 Разработка спецификации функциональных требований	33
2.6 Разработка технических требований к программному продукту	34
3 Проектирование архитектуры программного средства	35
3.1 Разработка архитектуры программного продукта.....	35
3.2 Проектирование алгоритмов ПС	39
4 Разработка программного средства.....	42
4.1 Выбор и обоснование языков программирования.....	42
4.2 Выбор среды разработки	44
4.3 Диаграммы классов программного средства	45
4.4 Описание компонентов клиентской части программного продукта	47
5 Тестирование программного средства.....	49
5.1 Методы тестирования	49
5.2 Тест-кейсы	50
5.3 Результаты тестирования	52
6 Руководство пользователя.....	55
6.1 Развёртывание сервера программного средства	55
6.2 Использование программного продукта.....	56
7 Технико-экономическое обоснование разработки интернет-магазина для ОАО «Світанак»	65
7.1 Описание функций, назначения и потенциальных пользователей программного продукта.....	65
7.2 Расчет затрат на разработку программного средства	65
7.3 Оценка экономического эффекта от использования ПС	68
Заключение	71
Список использованной литературы.....	72
Приложение А	73

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

Авторизация – предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

Аутентификация – проверка подлинности предъявленного пользователем идентификатора.

Инициализация – приведение областей памяти в состояние, исходное для последующей обработки или размещения данных.

Интерпретатор – программа или техническое средство, выполняющие интерпретацию.

Программа – данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма.

Программное обеспечение – совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ.

Программирование – научная и практическая деятельность по созданию программ.

Программный модуль – программа или функционально завершенный фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями и загрузки в оперативную память.

Спецификация программы – формализованное представление требований, предъявляемых к программе, которые должны быть удовлетворены при ее разработке, а также описание задачи, условия и эффекта действия без указания способа его достижения.

Фреймворк – программное обеспечение, облегчающее разработку и объединение различных компонентов большого программного проекта.

ООП – объектно-ориентированное программирование.

СУБД – система управления базами данных.

БД – база данных.

ПС – программное средство.

ПП – программный продукт.

ОС – операционная система.

Application Programming Interface (API) – интерфейс программирования приложений.

JavaScript Object Notation (JSON) – текстовый формат обмена данными.

Uniform Resource Locator (URL) – единообразный локатор ресурса.

HyperText Markup Language (HTML) - стандартизованный язык разметки веб-страниц в сети Интернет.

Cascading Style Sheets (CSS) – формальный язык описания внешнего вида веб-страницы.

Object-Relational Mapping (ORM) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования.

Model-View-Controller (MVC) – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер.

Lightweight Directory Access Protocol (LDAP) – протокол прикладного уровня для доступа к средству иерархического представления ресурсов.

OpenID – открытый стандарт децентрализованной системы аутентификации, предоставляющей пользователю возможность создать единую учётную запись для аутентификации на множестве не связанных друг с другом интернет-ресурсов.

ВВЕДЕНИЕ

Интернет-магазин – это прикладная система, построенная с использованием технологии электронной торговли. Подобно привычному представлению магазина, электронный магазин реализует следующие функции: представление товаров и их характеристик покупателю, обработку заказов, продажу и предоставление информации для дальнейшей доставки товаров покупателю.

Интернет-магазин объединяет элементы прямого маркетинга с образом посещения традиционного магазина. Отличительной чертой интернет-магазинов по сравнению с обычной формой торговли является то, что интерактивный магазин может предложить значительно большее количество товаров и услуг, и обеспечить потребителей значительно большим объемом информации, необходимым для принятия решения о покупке.

Основные проблемы реализации интернет-магазина лежат на стыке технологий Интернета и традиционной коммерческой деятельности. В обычной торговле покупатель привык к тому, что есть возможность оценить товар визуально, определить его качество и характеристики. В электронной торговле он такой возможности лишен. Иногда визуальной информации достаточно, но здесь имеет место быть эмоциональным и психологическим факторам.

По последним данным аудитория в интернете стремительно растет, а продажи через интернет в крупных городах достигают до 25%, при этом специалисты подчеркивают тенденцию к росту продаж именно через интернет. Ежегодно количество интернет-магазинов увеличивается, так как это действительно прибыльно и удобно для покупателя, не говоря о экономии бюджета и времени. Интернет-магазин работает круглые сутки и может продавать определенные товары в автоматическом режиме без участия продавца. К преимуществам так же можно отнести то, что не надо закупать товар заранее, а это существенная экономия, на складских помещениях. По сравнению с обычным магазином, территория продаж которого ограничивается территорией города или района, охвата интернет-магазина увеличивается на всю территорию Республики Беларусь и русскоязычную аудиторию в других странах, ведь товар можно доставлять не только курьерской службой, но и почтой.

Основной целью данной дипломной работы является разработка интернет-магазина по продаже трикотажных изделий из хлопка и смесей хлопка, выпускаемой ОАО «Світанак». Данная система будет работать в рамках определенного предприятия и выполнять функции дополнительной площадки для ведения коммерческой деятельности.

Дипломный проект выполнен самостоятельно и проверен в системе антиплагиат. Оригинальность составила 87,8% [1].

1 АНАЛИТИЧЕСКИЙ ОБЗОР ПРОГРАММНЫХ ПРОДУКТОВ, МЕТОДОВ И ПОДХОДОВ ПО ТЕМЕ ДИПЛОМНОГО ПРОЕКТИРОВАНИЯ

1.1 Основные понятия и определения в области электронной коммерции

Электронная коммерция – это форма поставки продукции, при которой выбор и заказ товаров осуществляется при помощи персонального компьютера или подобного ему устройства, а расчеты между покупателем и поставщиком осуществляются с использованием электронных документов и/или специальных средств платежа, при этом в качестве покупателей товаров или услуг могут выступать как частные лица, так и организации. Электронная коммерция представляет собой огромный комплект различных бизнес-операций.

Термин «электронная коммерция» объединяет в себе много различных технологий:

- электронный обмен информацией;
- электронное движение капитала;
- электронную торговлю;
- электронные деньги;
- электронный маркетинг;
- электронный банкинг;
- электронные страховые услуги.

Электронная торговля – осуществление торгово-закупочной деятельности через Интернет. Электронная коммерция – это общая концепция, содержащая в себе любые формы деловых операций, исполняемых электронным способом, а также использующая различные телекоммуникационные технологии. Деловые операции могут осуществляться напрямую между фирмами, фирмой и заказчиком, а также между фирмой и государственным учреждением.

Электронная коммерция реализуется в рамках интернет-экономики, которую нередко называют сетевой экономикой.

Сетевая экономика – среда, в которой каждая компания или индивид, находящиеся в любой точке экономической системы, могут контактировать свободно и с минимальными затратами с любой другой компанией либо индивидом по поводу совместной работы, для торговли либо для обмена идеями.

Современные информационные технологии, используемые в системе электронной коммерции, содержат в себе специальную инфраструктуру программного и аппаратного обеспечения, общие службы, специальные приложения, а также правовую структуру и соответствующие стандарты.

Основу функционирования системы электронной коммерции составляют электронные магазины.

Электронные магазины представляют собой реализованное предпринимателем представительство в сети Интернет на базе создания web-сервера. Основная цель создания такого предприятия состоит в обеспечении продажи товаров и оказании услуг другим пользователям сети Интернет.

Сфера применения системы электронной коммерции весьма разнообразны. Они охватывают широкий спектр коммерческих сделок (коммерческих сделок) и сделок, в частности:

- установить контакт между потенциальным покупателем и поставщиком;
- электронный обмен необходимой информацией;
- пред и послепродажная помощь покупателю, совершившему покупку в интернет магазине (предоставление подробной информации о продукте или услуге, предоставление инструкций по использованию продукта, оперативные ответы на вопросы клиентов
- осуществление прямого акта продажи товара или услуги;
- электронная оплата покупок (электронным переводом, кредитными картами, электронными деньгами, электронными чеками);
- предоставить покупателю товар, включая как управление отгрузкой, так и отслеживание физических товаров, а также прямую доставку товаров, которая может распространяться в электронном виде;
- создание виртуального предприятия, то есть группы независимых компаний, которые объединяют свои различные ресурсы, чтобы получить возможность предоставлять продукты и услуги, недоступные для независимых компаний;
- реализация независимых бизнес процессов, осуществляемых совместно производителем и его деловыми партнерами.

Электронная коммерция может осуществляться на двух уровнях: национальном и международном. Основанием для разницы в ведении бизнес операций на этих уровнях является не техническая и технологическая составляющие, а законодательная.

На международном уровне (по сравнению с национальным) внедрение системы электронной коммерции значительно усложняется. Это связано с такими факторами, как использование разных систем налогообложения, таможенных пошлин, принятие индивидуальных и в то же время неравноправных соглашений между разными странами, существенные различия в правилах проведения банковских операций.

Функционирование систем электронной коммерции на национальном уровне в основном связано с представлением компании в сети, рекламой, а также до и послепродажной поддержкой.

В системе электронной коммерции выделяют следующие категории организации коммерческой деятельности:

- коммерческие организации:
 1. Business to Business (B2B) – «взаимоотношения между коммерческими организациями»;

2. Business to Consumer (B2C) – «взаимоотношения между коммерческой организацией и потребителями»;
 3. Business to Employee (B2E) – «взаимоотношения между коммерческими организациями и сотрудниками (наёмными рабочими)»;
 4. Business to Government (B2G) – «взаимоотношения между организацией и правительством»;
- потребители:
1. Consumer to Administration (C2A) – «взаимоотношения между потребителями и администраторами»;
 2. Consumer to Business (C2B) – «взаимоотношения между потребителями и коммерческими организациями»;
 3. Consumer to Consumer (C2C) – «взаимоотношения между потребителями»;
- администрация:
1. Administration to Administration (A2A) – «взаимоотношения между администрациями»;
 2. Administration to Business (A2B) – «взаимоотношения между администрацией и коммерческими организациями»;
 3. Administration to Consumer (A2C) – «взаимоотношения между администрацией и потребителями».

1.2 Обзор тенденций в области Интернет-торговли

Объем продаж в области электронной торговли по данным статистической компании «Statista» за 2019 год достиг 3,5 триллиона долларов, что является почти 4% мирового ВВП. Однако недавний кризис оказал в некотором смысле положительное влияние на легкую промышленность.

Объем продаж, осуществляемый посредством сети Интернет показал быстрый темп роста и в 2020 году достиг 4,2 триллиона долларов. По некоторым прогнозам, придерживаясь такого темпа роста (20% в год), к 2023 году он составит порядка 6,5 трлн долларов, что в эквиваленте ВВП составит уже почти 7% [8], а к 2036 году объемы мирового рынка eCommerce превысят объемы традиционной розницы. Диаграмма роста объема продаж, демонстрирующая данное увеличение, отображена на рисунке 1.1.

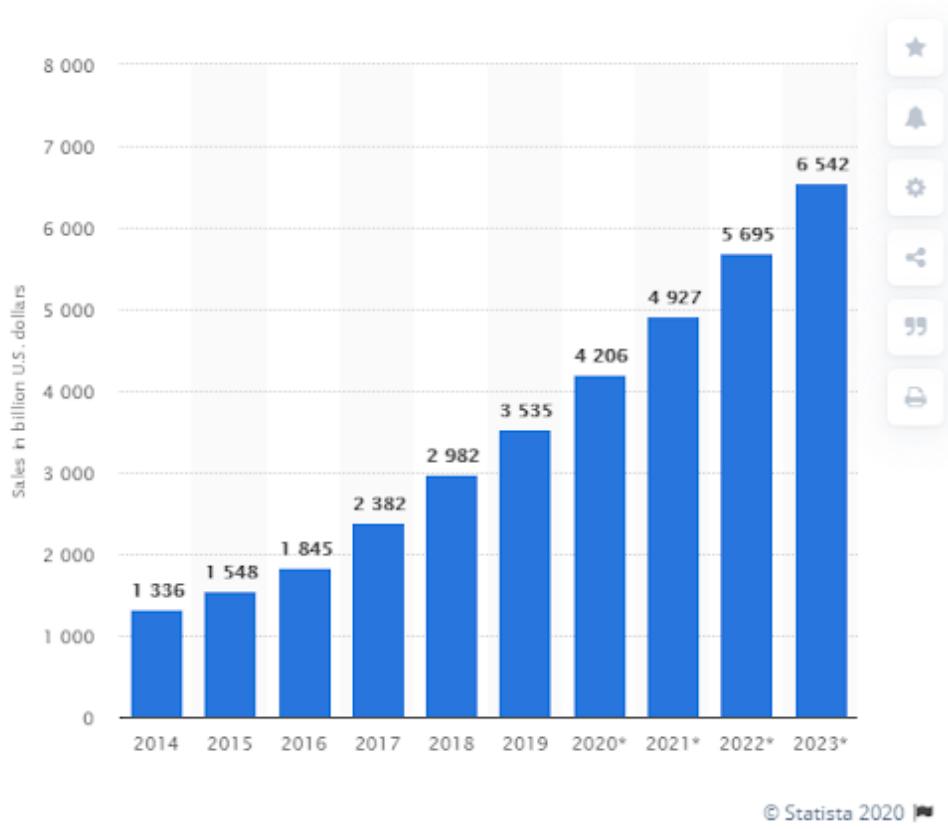


Рисунок 1.1 – Диаграмма роста объема продаж в области электронной коммерции от статистической компании «Statista»

Попадая на веб-сайт пользователи в течение некоторого времени формируют мнение о данной платформе. Разработка программного средства в соответствии с текущими тенденциями позволяет создать продукт, удовлетворяющий требования большинства покупателей.

Порядка 50% всех покупок в интернет-магазинах и чуть менее половины транзакций происходит с мобильных устройств. И доля подобных покупок и транзакций продолжает расти. Поэтому обязательными функциями для удобства пользователей являются лёгкая навигация, быстрая и простая процедура оформления заказа вкупе с несколькими вариантами оплаты заказа.

Преимущество адаптивного дизайна заключается в том, что вам не нужно создавать совершенно другую мобильную версию для вашего сайта. Адаптивный дизайн адаптируется к соответствующему устройству, и единственное, что вам нужно проверить, - это то, что изображения и размеры отображаются правильно.

Следующим пунктом можно рассмотреть такой тип дизайна, как минималистичный, либо же плоский веб-дизайн. Большим преимуществом такого дизайна является то, что он чистый и простой. Основное внимание в этих проектах уделяется удобству восприятия информации и простоте навигации. Минимализм – это одна из основных тенденций в электронной коммерции. Обесцвечивая интерфейс, делая его нейтральным, дизайнеры отдают главную роль

фотографиям товара, высококлассным рендерам, исчерпывающему описанию и понятному уникальному торговому предложению.

Интерфейс должен выполнять исключительно служебную функцию, объединять пользователя с сайтом и позволять ему осуществлять ключевые действия. Здесь основным является контент, а интерфейс может повлиять на принятие решения, взять на себя акцент.

Типографика. В прошлом шрифты и типографика не были важной частью, рассматриваемой при проектировании интернет-магазинов. Но оказалось, что иногда шрифты и текст могут говорить громче, чем фотографии или видео. Это философия больших букв и массивного текста. Большие буквы впечатляют и наверняка привлекают внимание людей. В то же время сайт содержит основную информацию, которая нужна покупателям - информацию о товаре, корзину, меню.

Оптимизация логистики. В США набирает популярность модель, когда доставка идет до ближайшего магазина у дома. Интернет-магазинам необходимо искать методы минимизации своих расходов на доставку. Поэтому логистика, с большой вероятностью, будет в формате доставки к магазинам у дома.

Сектор B2B. Электронная коммерция кардинально изменила способ покупки в традиционном виде и онлайн. Развивающиеся программные инновации, такие как ориентация на предоставление персонализированных покупок, улучшают комфорт клиентов с каждым днем. Наряду с этим их ожидания также растут. Следовательно, если необходимо эффективно привлекать своих клиентов, владельцы интернет-магазинов никогда не должны прекращать обновление своего магазина, чтобы удовлетворить их требования.

1.3 Анализ существующих программных решений по теме дипломного проектирования

Наличие высокого спроса на покупку товаров в режиме онлайн привело к тому, что большинство крупных магазинов либо имеют площадку для ведения электронной торговли, либо же планируют ее создание. На данный момент число таких сервисов не ограничивается одним десятком, тематика которых варьируется от продажи гаджетов до оказания различного рода услуг, так что выберем максимально похожие – Conte Shop, Veore.by и Купалинка.

1.3.1 Conte Shop

Одной из самых известных площадок белорусских производителей, которые позволяют приобретать в режиме онлайн свою продукцию является компания «Конте Шоп» [13]. Его популярность обусловлена целым рядом факторов, среди которых и большой ассортимент предлагаемой продукции, и удобный сервис для демонстрации, и предоставления возможности заказа товаров.

На рисунке 1.2 представлен одна из страниц, на которую попадает пользователь для просмотра конкретной категории каталога.

The screenshot shows the website for 'Conte FAMILY COLLECTION'. At the top, there are links for 'Минск' (Minsk), 'Магазины' (Stores), and a phone number '+375 (29) 622-69-66'. The navigation menu includes 'ЖЕНЩИНАМ' (Women), 'МУЖЧИНАМ' (Men), 'ДЕВОЧКАМ' (Girls), 'МАЛЬЧИКАМ' (Boys), and 'АКЦИИ' (Promotions). On the right side, there are links for 'Подарочные карты' (Gift Cards), 'Как заказать' (How to Order), 'Доставка' (Delivery), 'Отзывы' (Reviews), 'Возврат и гарантия' (Return and Warranty), and a user icon. A search bar with the placeholder 'Искать по сайту' (Search site) is located at the top right. Below the header, a breadcrumb navigation shows 'Домой / Мужчинам / Джемперы / С длинным рукавом'. The main content area displays a grid of three men's turtleneck sweaters in dark blue, white, and maroon. Each item has a small image, the product name 'Джемпер MD 816', and the price '35,99 руб.'.

Домой / Мужчинам / Джемперы / С длинным рукавом

С длинным рукавом для мужчин

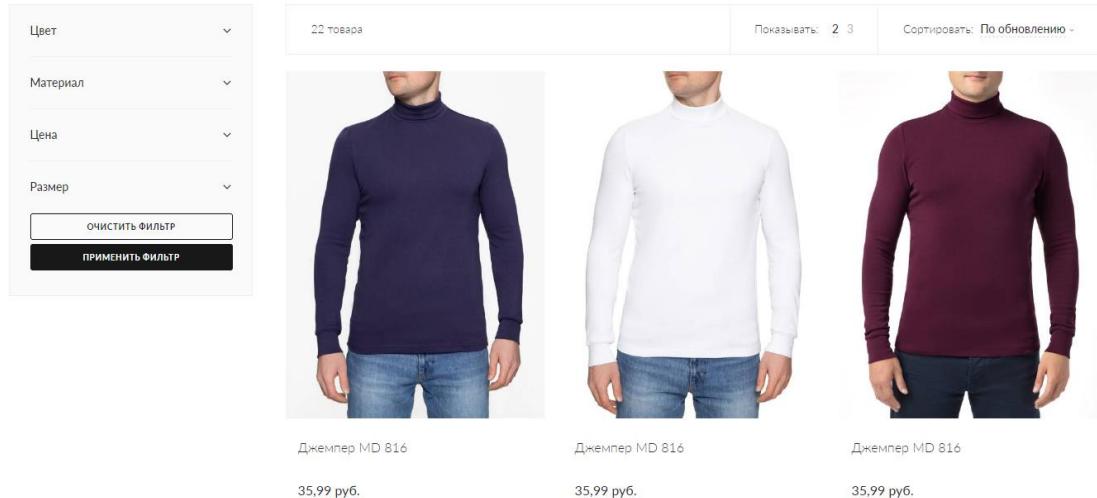


Рисунок 1.2 – Каталог интернет-магазина «Конте Шоп»

Интернет-магазин «Конте Шоп» предоставляет потенциальному покупателю следующую информацию:

- список товаров выбранной категории с наименованием товара и его ценой;
- количество товаров в категории;
- возможность быстрого просмотра карточки товара;
- возможность выбрать количество товаров в блоке;
- возможность сортировки;
- фильтр по таким параметрам, как цвет товара, материал, цена и размер.

Карточка товара, изображенная на рисунке 1.3, приветствует покупателя большим слайдером с фотографиями выбранного товара, а также возможностью более детально просмотреть фото. Справа от слайдера отображен блок с ценой, допустимыми цветовыми решениями данной модели, доступными комбинациями размера-роста, возможностью выбрать сразу несколько одинаковых товаров для добавления в корзину, а также кнопкой для добавления товара в список желаемых



Рисунок 1.3 – Страница определенного товара интернет-магазина «Конте Шоп»

При нажатии кнопки «Добавить в корзину» появляется попап-блок, который отображает добавленный в корзину товар и предоставляет покупателю выбор между возможностью продолжения покупки и переходом к оформлению заказа. Данное всплывающее окно отображено на рисунке 1.4.

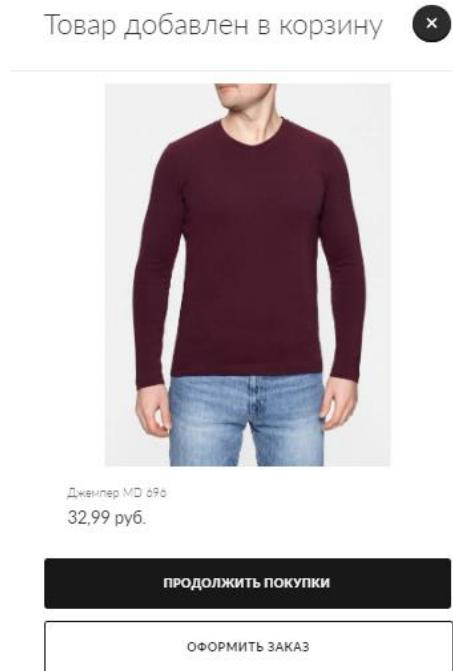


Рисунок 1.4 – Всплывающее окно подтверждения совершения действия

Выбрав вариант «Продолжить покупки» всплывающее окно скрывается и нам снова предоставляется возможность просмотра каталога.

При переходе в корзину покупателю предоставляется:

- возможность просмотреть список выбранного товара;
- возможность изменить количество определенных позиций в списке;
- возможность ввода номера дисконтной карты или кода, предоставляющего скидку;
- сумма к оплате.

Страница с содержимым корзины покупок отображена на рисунке 1.5.

The screenshot shows the shopping cart page of the Conte Shop website. At the top, there is a navigation bar with the Conte logo, categories for ЖЕНЩИНАМ, МУЖЧИНАМ, ДЕВОЧКАМ, МАЛЬЧИКАМ, and АКЦИИ, a search bar, and user icons for login, favorite items, and the shopping cart. The shopping cart itself contains two items: two dark red crew-neck sweaters (Джемпер MD 696) priced at 32,99 rub. each. To the right of the cart, there is a sidebar with a discount input field ('Введите номер Вашей дисконтной карты или код на скидку, если он у вас есть'), a 'Применить' button, and delivery information ('Доставка и обработка (БелПочта - До отделения Белпочты) 0,00 руб.'). At the bottom right, it shows the total amount 'Итого к оплате 65,98 руб.' and a large red 'ОФОРМИТЬ ЗАКАЗ' button.

Рисунок 1.5 – Содержимое корзины и оформление заказа в интернет-магазине «Конте Шоп»

Одной из важных отличительных черт данного сервиса можно назвать строгое выдержанное оформление, как элементов каталога, так и элементов всего сайта, которое не загромождено большим количеством ненужной информации. Мобильная версия сайта также выполнено в общей стилистике, но имеет ряд недостатков, такие как отсутствие возможности увеличения фотографии без перехода на новую страницу, трудночитаемый шрифт, обусловленный малым размером символов, наличие всплывающих окон, в которых запрашиваются личные данные.

Несмотря на то, что данный сервис обладает большим количеством достоинств, стоит отметить и ряд общих недостатков:

- отображение товаров при отсутствии возможности добавления в корзину заказа;
- отсутствие фото для некоторых цветовых вариаций товара;

- отсутствие информирования о доступном для заказа количестве единиц товара;
- отсутствие детальной информации о добавленном в корзину товаре;
- наличие всплывающих окон с просьбой предоставления адреса электронной почты для последующих рекламных рассылок.

В мобильной версии сайта также было выявлено несколько недостатков: это отсутствие возможности увеличения фотографии без перехода на новую страницу и трудночитаемый шрифт, обусловленный малым размером символов.

1.3.2 Veore.by

Еще одним рассматриваемым решением, существующим в данной предметной области, является интернет-магазин компании «Veore» [12]. Данный сервис ориентирован на продажу товара разным возрастным группам. В таких направлениях как одежда женщинам, мужчинам, детям, красота, товары для дома и гаджеты имеется более двух тысяч позиций. На рисунке 1.6 отображена предлагаемая продукция одной из возможных категорий.

Новые товары Написать нам Доставка Новые товары Написать нам Доставка Войти

veore - It's a woman's world

Новые товары Написать нам Доставка Новые товары Написать нам Доставка Войти

Все категории Поиск КОРЗИНА: (пусто)

ЖЕНЩИНАМ МУЖЧИНАМ ДЕТИМ КРАСОТА ДЛЯ ДОМА ГАДЖЕТЫ СКИДКИ

Мужчинам Одежда Рубашки

ЦЕНА

Диапазон: 13,00руб. - 35,00руб.

РАЗМЕР

S (2)
M (3)
L (4)
XL (2)
XXL (3)
XXXL (1)

СЕЗОН

весна/осень (1)
лето (8)

РУБАШКИ

Показать 1 - 11 из 11 товаров Пор Сортировать по

11 товаров.

РУБАШКА МУЖСКАЯ Y3664 СИНИЙ 17,60 руб. 22,00 руб.	РУБАШКА МУЖСКАЯ Y3667 БЕЛЫЙ 22,00 руб.	МАЙКА Y3675 ЧЕРНЫЙ 13,50 руб. 18,00 руб.	МАЙКА Y3680 СИНИЙ 18,00 руб.
МАЙКА Y3680 СЕРЫЙ НЕТ В НАЛИЧИИ	КОФТА Y3676 СИНИЙ НЕТ В НАЛИЧИИ	МАЙКА Y3680 БЕЛЫЙ НЕТ В НАЛИЧИИ	МАЙКА Y3675 БЕЛЫЙ/СИНИЙ НЕТ В НАЛИЧИИ

veore/by
SALE
ЦЕНЫ НИЗКИЕ
КАК НИКОГДА!
проверь

ПРОСМОТРЕННЫЕ ТОВАРЫ

	ШТАНЫ Y5936 СИНИЙ
	ЖИЛЕТКА СИНИЙ Y4701 НЕТ В НАЛИЧИИ

Рисунок 1.6 – Категория «Рубашки» интернет-магазина «Veore»

Самое первое отличие, которое бросается в глаза после рассмотрения интернет-магазина компании «Conte», является то, что данный сервис предполагает не только продукцию легкой промышленности, но и товары таких категорий как «Красота» и «Гаджеты».

Второе отличие, которое становится очевидным при пользовании сервисом – это отсутствие или неработоспособность некоторых привычным всем функций. Фильтр товара в каталоге осуществляется только по трем параметрам: диапазон цен, размер и сезон, а просмотр характеристик возможен только при переходе на страницу конкретного товара. Блок, позволяющий определить количество отображаемых на одной странице товаров некорректно выровнен.

Большим плюсом является то, что фильтр реализован с применением технологии Ajax, которая предоставляет возможность изменять отображение товаров при активных позициях фильтра без перезагрузки страницы. Рядом с названием элемента фильтра размещена информация о количестве товаров с одноименным параметром.

После перехода на страницу определенного товара пользователю предоставляется возможность просмотра такой информации, как допустимые размеры, наличие товара на складе в текущий момент, кнопка «Добавить в корзину», производитель, а также дополнительную информацию из разделов «параметры», «информация» и «возврат». Результат перехода изображен на рисунке 1.7.

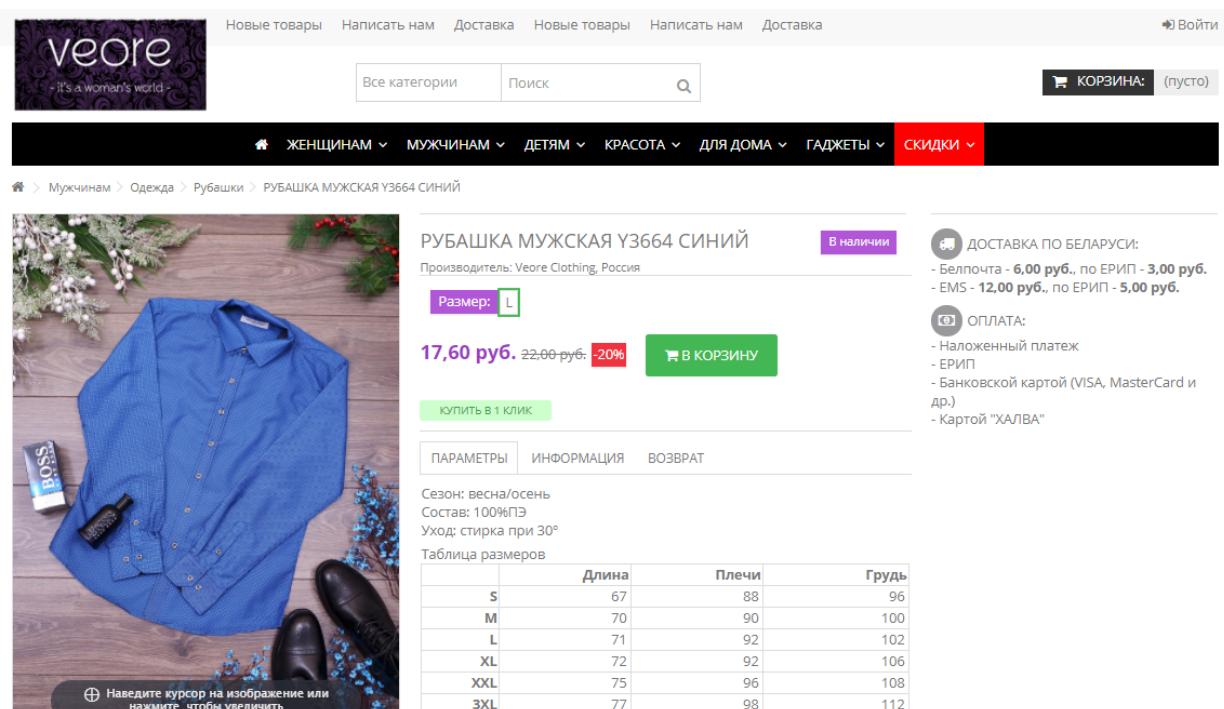


Рисунок 1.7 – Страница с детализацией товара

Отсутствие фильтрации по цвету обусловлено наличием определения цвета в названии товара, что нельзя трактовать как недостаток, хотя это и препятствует поиску товаров по определенному цвету, т.к. глобальный поиск по сайту отобразит товары из всех категорий.

Таблица размеров является уникальной для каждого товара, определяемой на основании принадлежности товара к определенной категории и полу.

После нажатия кнопки «Добавить в корзину» пользователю отображается всплывающее окно, гласящее об успешном выполнении операции и предлагает два варианта продолжения работы: продолжить покупки и перейти к оформлению заказа. Данное окно отображено на рисунке 1.8. Нажатие на кнопку продолжения также не перезагрузит страницу, что положительно скажется на работе покупателя.

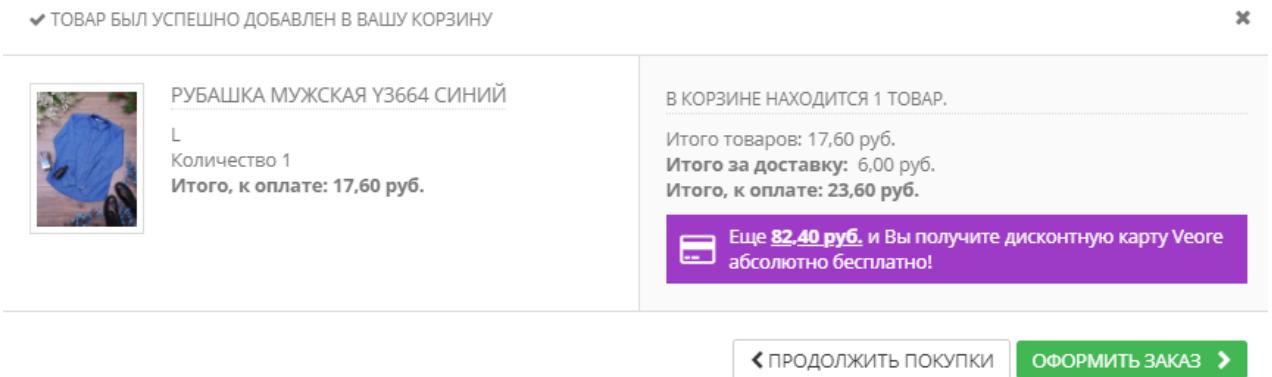


Рисунок 1.8 – Всплывающее окно при добавлении в корзину

Перейдя к оформлению заказа пользователю будет отображена страница, содержащая содержимое корзины, выбор способа доставки и оплаты, а также данные аккаунта. Дополнительными функциями являются возможность удаления товара из списка и применение скидочного промокода.

Увеличение количества определенного товара влечет за собой ошибку, текст которой отображается с помощью функции вызова модальных окон `alert()`, что тоже негативно сказывается на работе с данной платформой. При условии сбора корзины не авторизованным пользователем нам будет предложено пройти процесс авторизации либо регистрации, введя данные в соответствующие поля и нажав кнопку «Оформить заказ», предварительно установив флажок в чекбокс «Регистрация?», что не является интуитивно понятным. Это изображено на рисунке 1.9.

veore.by says
Не достаточно товара в наличии.

OK

Войти

КОРЗИНА: 1 Товар: 17,60 руб.

ЖЕНЩИНАМ

СКИДКИ

Оформление заказа

АККАУНТ		ЗАРЕГИСТРИРОВАНЫ?
Фамилия Имя Отчество: *	<input type="text"/>	
Адрес покупателя (только Беларусь): *	<input type="text"/>	
Индекс покупателя:	<input type="text"/>	
Комментарий:	<input type="text"/>	
Телефон:	<input type="text"/>	
Ваше имя: *	<input type="text"/>	
Ваша фамилия: *	<input type="text"/>	
E-mail:	<input type="text"/>	
<input type="checkbox"/> Регистрация?		
Имя и фамилия менеджера:	<input type="text"/>	

Методы доставки

<input checked="" type="radio"/> БЕЛПОСТ	Обычное отправление	Доставка в течении 2-3 дней.	6,00 руб.
<input type="radio"/> EMS BELPOST	Ускоренная почта	EMS На следующий день после заказа.	12,00 руб.
<input type="radio"/> Курьером по Гродно	Лучшая скорость	По рабочим дням.	5,00 руб.

Методы оплаты

<input checked="" type="radio"/> Наложенным платежом (при получении почтового отправления)
<input type="radio"/> Банковской картой (VISA, Mastercard, ХАЛВА и др.)
<input type="radio"/> Оплата ЕРИП. Экономия: -3,00 руб..
<input type="radio"/> Оплатить бонусами (Недостаточно денег)

Содержимое вашей корзины: 1 товар(ов)

Товар	Описание	Цена за единицу	Кол-во	Всего
	РУБАШКА МУЖСКАЯ Y3664 СИННИЙ Размер : L	17,6 руб. -20% 22,00 руб.	<input type="button" value="1"/> <input type="button" value="-"/> <input type="button" value="+"/>	17,6 руб.
Введите код и получите скидку		<input type="text"/>	<input type="button" value="OK"/>	
		Tоваров	17,6 руб.	
		Доставка	6,00 руб.	
		Всего	23,6 руб.	

Доставка осуществляется только по Беларуси.

ОФОРМИТЬ ЗАКАЗ

Рисунок 1.9 – Страница оформления заказа

Подведя итоги анализа данного интернет-магазина можно выявить следующие недостатки:

- неработоспособность некоторых функций;
- отсутствие в фильтре множества параметров товаров;
- отсутствие возможности выбрать определенное количество товара на странице самого товара, изменение которого производится только в корзине, а попытки изменения приводят к возникновению ошибки;
- уведомление посредством модальных окон.

1.3.3 ОАО "Купалинка"

Очередным сервисом, существующим в данной предметной области, является интернет-магазин компании «Купалинка» [14].

Сервис предлагает к продаже бельевой и верхний трикотаж для взрослых и детей. По способу взаимодействия с пользователем интернет-магазин имеет сходства с Conte Shop: интерфейс минималистичен, в каждый момент времени отображаются только самая необходимая информация. Такой способ

взаимодействия с пользователем позволяет сконцентрировать внимание на подробном изучении информации о товарах.

На рисунке 1.10 представлен раздел «Мужчинам» каталога продукции.

ЖЕНЩИНАМ МУЖЧИНАМ ДЕВОЧКАМ МАЛЬЧИКАМ МАЛЫШАМ НОВИНКИ МАСКИ

Главная - Каталог - МУЖЧИНАМ

МУЖЧИНАМ

Сортировать По названию ▲ По цене ▲ По популярности Нет Вид

ЖЕНЩИНАМ 144
МУЖЧИНАМ 51
ДЕВОЧКАМ 58
МАЛЬЧИКАМ 26
МАЛЫШАМ 32
НОВИНКИ
МАСКИ

Розничная цена
От 0 До 255 407
Состав полотна
Цвет
Размеры одежды

Показать Сбросить

РОДНАЯ БЕЛАРУСЬ ДЖЕМПЕР МУЖСКОЙ 180302 23.30 руб

ДЖЕМПЕР МУЖСКОЙ 258007 22.39 руб

ТРУСЫ МУЖСКИЕ 226824 11.70 руб

ФУТБОЛКА МУЖСКАЯ 260007
МАЙКА МУЖСКАЯ 254504
ФУТБОЛКА МУЖСКАЯ 2112BW

Рисунок 1.10 – Раздел «Мужчинам» интернет-магазина «Купалинка»

Возможности данного сайта весьма обширны, пользователь может выбрать конкретную категорию для просмотра, воспользоваться фильтром, который поможет найти необходимые товары при помощи указания диапазона цен, состава полотна, цвета и размера желаемого продукта. Также имеется возможность сортировки по названию, цене и популярности как от меньшего к большему, так и наоборот, функции «Быстрый просмотр», «Купить в один клик» и добавить товар в список желаемого. Изменяя вариант отображения товаров сайт преображается, что позволяет совершать меньше действий для просмотра таких параметров модели, как диапазон размеров и доступные цвета модели.

Воспользовавшись быстрым просмотром, покупателю отображается всплывающее окно, которое отображено на рисунке 1.11.



ДЖЕМПЕР МУЖСКОЙ 180302

23.30 руб

В корзину



X



Определите свой размер

Подробнее

Рисунок 1.11 – Быстрый просмотр товара

Как можно заметить, быстрый просмотр не позволяет в полной мере оценить доступные размерные и цветовые вариации товара, что в некоторой мере сковывает действия покупателя. Клик на картинку либо наименование товара адресует покупателя на страницу товара, в которой можно просмотреть параметры модели и добавить необходимый в корзину. Описание модели строится на основании четырех атрибутов: наименования модели, артикула, цвета и размера.

Добавив необходимое количество товаров, пользователь переходит в корзину для оформления заказа. Страница корзины отображена на рисунке 1.12. Из доступного функционала можно отметить возможность изменения количества определенного товара, автоматический подсчет суммы заказа, поле для ввода кода купона, возможность добавить товар в список желаемого и удалить из корзины необходимые позиции. Можно заметить неровное отображение блока с товарами, который искусственно выровнен по правому краю, что создает большое количество неиспользуемого пространства слева от последнего.

КОРЗИНА

Готовые к заказу

Товары	Цена	Количество	Сумма		
 ФУТБОЛКА МУЖСКАЯ 182003_88 Цвет: морской Размеры одежды : 44	20.44 руб	- 1 +	20.44 руб		
 ДЖЕМПЕР МУЖСКОЙ 180302	23.30 руб	- 1 +	23.30 руб		
 ФУТБОЛКА МУЖСКАЯ 260007_46 Цвет: серый Размеры одежды : 46	15.80 руб	- 1 +	15.80 руб		
 Майка мужская 132228_88 Цвет: белый Размеры одежды : 44	8.70 руб	- 1 +	8.70 руб		

Введите код купона для скидки:

Итого: 68.24 руб

[Оформить заказ](#)

Рисунок 1.12 – Корзина интернет-магазина «Купалинка»

Следующим шагом, который совершает пользователь для заказа является переход к оформлению заказа. Даже авторизованному пользователю сразу же отображается форма для ввода информации о покупателе, среди которых имеются такие поля как название компании, юридический адрес, контактное лицо, что уже смущит подавляющее большинство покупателей, ведь нигде нет информации о том, что интернет-магазин является B2B-ориентированным, что подтверждается еще и наличие только одного вида оплаты заказа – наличными.

Детализация заказа, а также возможность написать комментарий к нему размещены снизу от формы для ввода персональной информации.

Несмотря на высокий показатель удобства пользования, данный сервис не обделен недочетами, основными из которых являются:

- некорректно работающие функции сортировки;
- отсутствие ограничения цен в фильтре;
- искажение положений блоков;
- некорректное отображение текста на большинстве страниц при просмотре с мобильного устройства.

1.4 Постановка целей и задач дипломного проектирования

Исходя из результатов анализа предметной области, основными недостатками существующих решений являются следующие:

- наличие недоработок мобильных версий сайта;
- отсутствие детализации предлагаемых продуктов;
- недостаточный уровень удобства пользования;
- медленная работа сайта, отображение данных с задержками;
- отсутствие обмена данными при работе с нескольких устройств одновременно.

Исходя из всего вышеупомянутого, целью дипломной работы является создание интернет-магазина с гибким функционалом, понятным и доступным интерфейсом, возможностью дальнейшего расширения каталога продукции, а также путем добавления новых функций. Данная система будет выполнять функции площадки для ведения дополнительной коммерческой деятельности предприятия.

Реализация поставленной цели предполагает решение следующих задач:

- выбор средства разработки интернет-магазина;
- разработка интернет-магазина в соответствии со структурой базы данных;
- отображение каталога продукции компании, включая такие поля, как наименование, цена, размерная сетка, состав сырья, допустимые цветовые вариации и наличие, либо же отсутствие изделий на складе;
- осуществление сортировки каталога продукции;
- осуществление возможности поиска по нескольким параметром продукта;
- осуществление регистрации и авторизации пользователей, а также разграничение прав между ними;
- создание зарегистрированным пользователем заявки на покупку собранной корзины, а также выбор способа оплаты и доставки;
- осуществление обратной связи с пользователем через E-mail.

После создания программного средства необходимым является проведение тестирование, написание руководства пользователя и создание инструкции по развертыванию сервера программного средства.

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ И СОСТАВЛЕНИЕ ИХ СПЕЦИФИКАЦИИ

2.1 Общие сведения и требования к работе программного продукта

Функциональным назначением разрабатываемого программного решения является предоставление пользователю возможностей для проведения онлайн покупок.

Пользователем в данном случае может выступать любой человек, который имеет доступ к сети Интернет. Для использования программного средства не требуется специальная подготовка или обучение пользователей.

Предполагается возможность одновременной эксплуатации разрабатываемого программного продукта большим числом пользователей. При этом отсутствуют какие-либо ограничения, накладываемые на предметную область, в рамках которой возможно его применение.

Исходя из предполагаемого использования, можно заключить, что проектируемое программное решение должно реализовывать следующие три группы функций:

- управление каталогом, в частности товарами и их параметрами;
- управление результатами посещений пользователями, т.е. непосредственная обработка заказов;
- управление пользователями.

2.2 Описание функциональности программного продукта

Средством представления функциональности программного средства будет выступать диаграмма вариантов использования. Настоящий вид UML диаграмм позволяет описать функциональность системы на концептуальном уровне посредством построения взаимосвязей между двумя основными элементами: прецедентами и актерами.

Каждый прецедент Use-Case диаграммы отображает один из вариантов использования программного средства конкретным пользователем. Средства UML позволяют установить отношения обобщения для актеров на диаграмме, таким образом отсутствует необходимость в дублировании одинаковых прецедентов на диаграмме использования.

На рисунке 2.1 представлена обобщенная диаграмма вариантов использования разрабатываемого программного средства, на которой отражены группы функций, доступные для действующих лиц внутри рамок проектируемой системы.

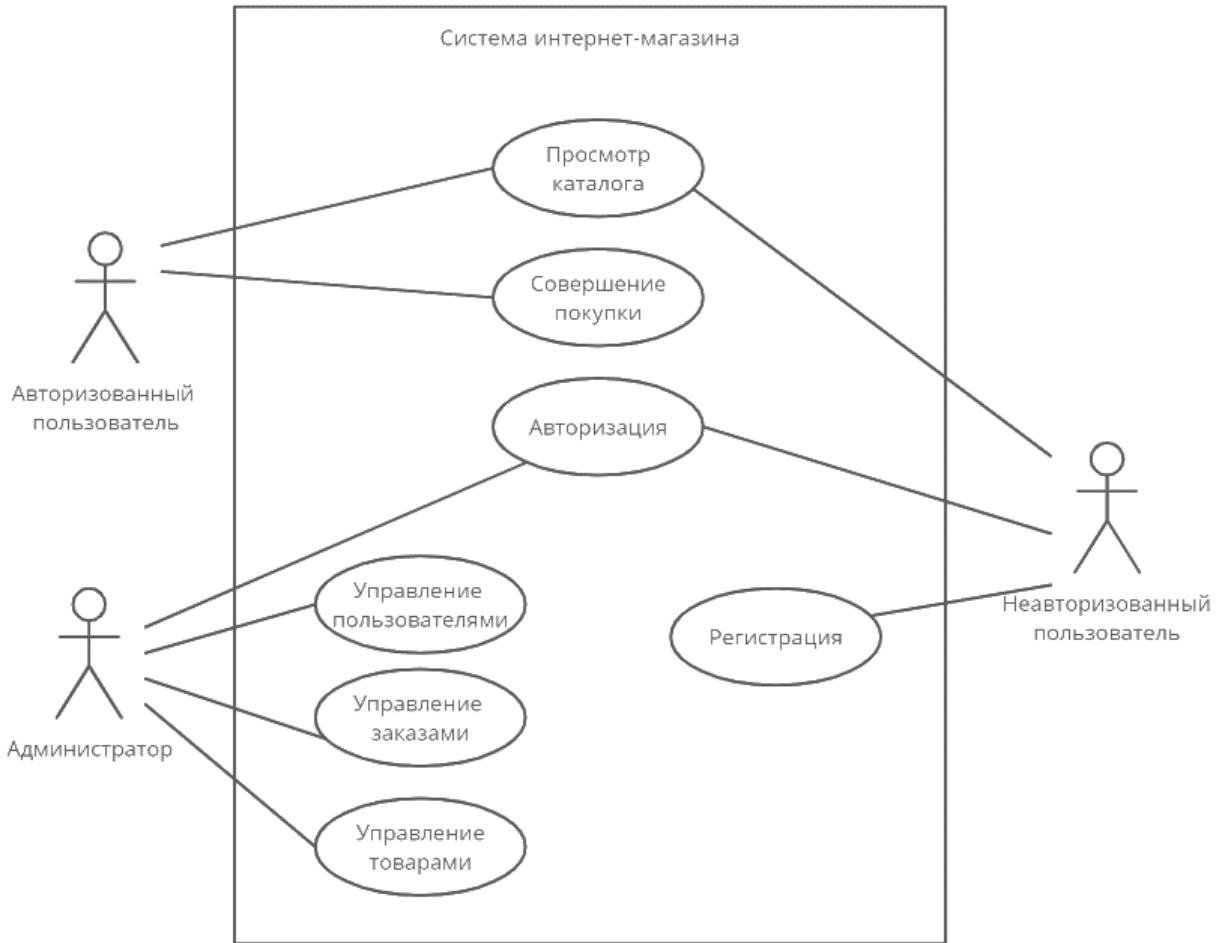


Рисунок 2.1 - Обобщенная диаграмма вариантов использования

В процессе анализа существующих программных решений в данной предметной области был определён список действующих лиц и ролей, которые будут взаимодействовать с разрабатываемой системой, и для которых, в свою очередь, данная система должна будет предоставлять ряд функций. Среди таких действующих лиц мы можем выделить следующие:

- неавторизованный (анонимный) пользователь;
- авторизованный пользователь;
- администратор.

Рассмотрим всех действующих лиц проектируемого решения в порядке их следования. Анонимный пользователь – базовая роль для остальных актеров на диаграмме прецедентов, к данному типу мы относим всех пользователей, о которых разрабатываемое решение не имеет никакой информацией, которая бы позволила идентифицировать в них зарегистрированного пользователя. Однако стоит отметить, что данный тип действующих лиц является одним из основных в рамках программного средства, а многие из пользователей, которые используют системы подобного типа, не имеют персонального аккаунта.

На рисунке 2.2 отображены функциональные возможности анонимного пользователя.

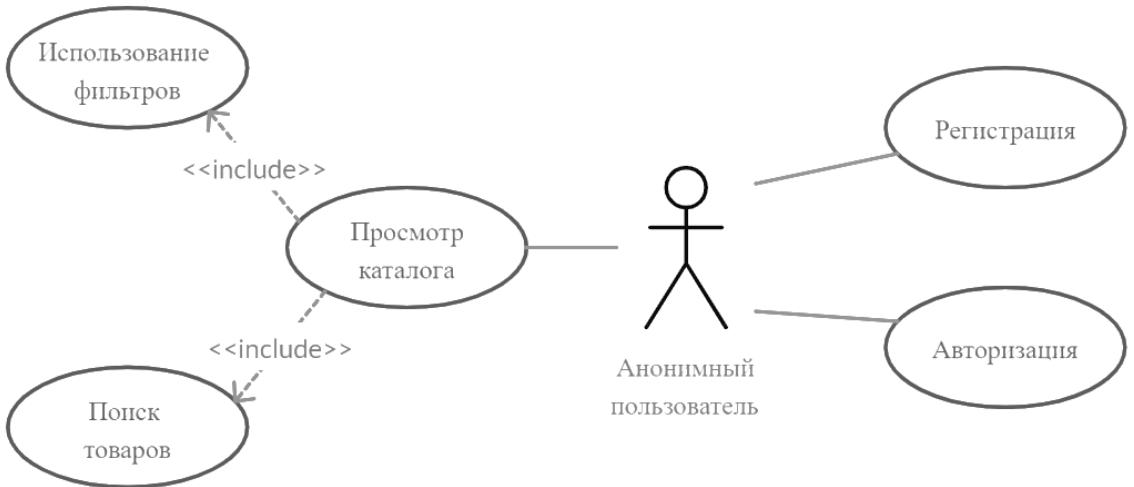


Рисунок 2.2 – Функциональные возможности анонимного пользователя

Наличие отдельного пользовательского профайла обусловлено тем, что для правильного функционирования системы необходимо некоторое количество пользовательских данных. Для этого есть несколько причин: возможность формирования заказа предусматривает бронирование определенного количества товаров для пользователя, совершение оплаты и доставка также не осуществимы по той же причине.

Данному типу пользователей предоставляются следующие функции в рамках системы:

- регистрация;
- авторизация;
- просмотр каталога товаров.

Регистрация пользователя в системе подразумевает создание персонального аккаунта, а также предоставление пользователям прав роли типа «Пользователь». В процессе регистрации пользователю необходимо ввести персональные данные среди которых: адрес электронной почты, пароль, ФИО и город. В последствии адрес электронной почты и персональный пароль будут использоваться при осуществлении идентификации пользователей системы.

Процессом авторизации будем называть идентификацию пользователя по предоставленным адресу электронной почты и персональному паролю, зарегистрированного в системе пользователя и дальнейшее предоставление ему прав доступа в соответствии с занимаемой в системе ролью.

Основной функцией, предоставляемой программным решением для пользователей с данным типом роли является возможность просмотра каталога товаров, разделенных на категории, фильтрация отображаемых данных посредством ввода определенных критериев отображения данных, а также возможность использования поиска.

Следующим видом актера, изображенным на диаграмме прецедентов, является авторизованный пользователь. Данный тип роли наследует все функциональные возможности анонимного пользователя, но кроме уже рассмотренных функций, пользователю также предоставляется возможность управления личными данными, формированием заказа и выбором способов оплаты и доставки.

На рисунке 2.3 детально отображены возможности, которые доступны для пользователей с данной ролью.

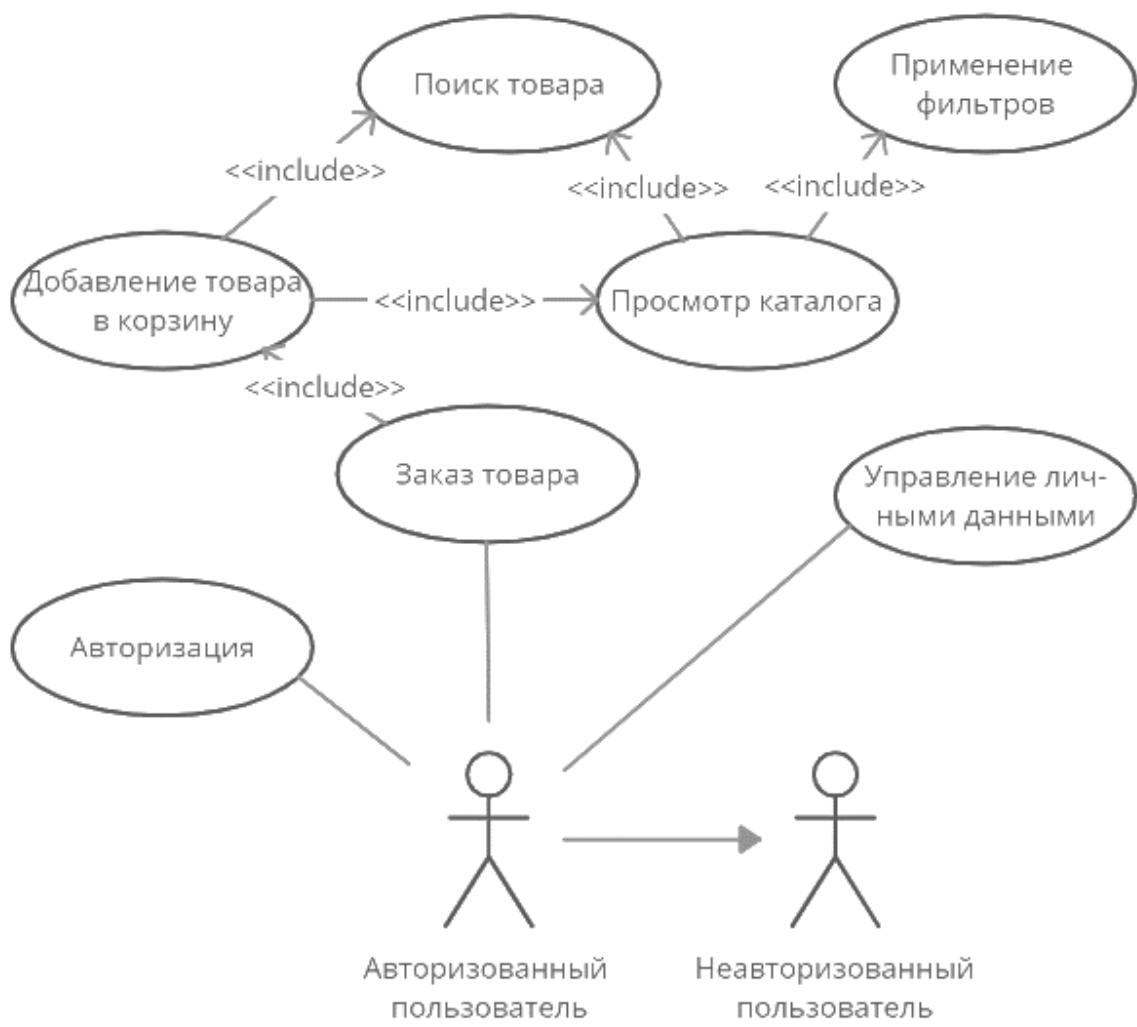


Рисунок 2.3 – Возможности авторизованного пользователя

Основной функцией, предоставляемой программным решением является возможность осуществления заказа товаров из собранной пользователем корзины. Данная функция подразумевает открытие каталога товаров, осуществление логики переходов между страницами в процессе работы пользователя с программным средством, формирование корзины, которая основывается на выбранных пользователем товарах в процессе работы с сервисом, сохранение данных по окончанию процесса поиска и выбора товаров, а также оформление заказа.

В процессе формирования заказа должны осуществляться все необходимые проверки введённых и выбранных пользователем значений, осуществляться отображение элементов управления, а также следует обеспечить корректное функционирование переходов между отдельными страницами сайта.

Последним типом пользовательских ролей, представленных на диаграмме прецедентов, является администратор. Администратор – тип роли, предназначенный для администраторов системы. Пользователи с данной ролью должны обладать максимальным набором функций из списка возможных. Администратор включает в себя возможности остальных типов ролей. Основное предназначение данной роли – администрирование системы в целом и контроль над всеми процессами, происходящими внутри системы. Исходя из этого, пользователи данного типа агрегируют возможности остальных ролей. При развертывании программного средства в системе должен существовать как минимум один пользователь с ролью данного типа.

Полный список возможностей, внесенных для пользователей данного типа, изображен на рисунке 2.4.

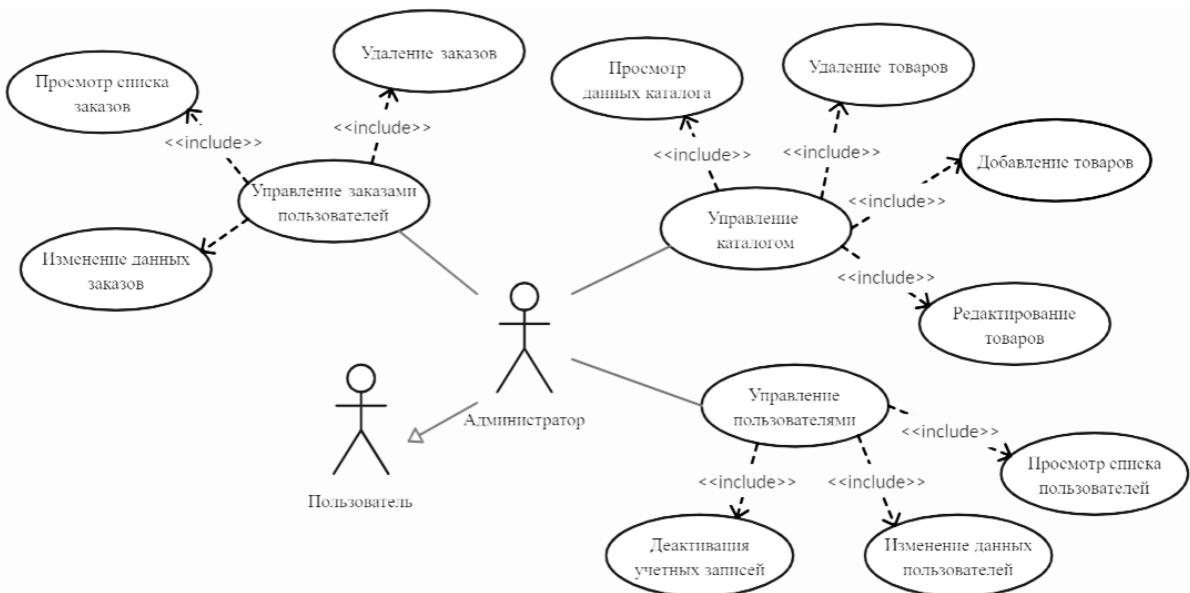


Рисунок 2.4 – Функциональные возможности администратора

Для обеспечения работы сервиса важно знать количество существующих пользователей, иметь возможность просмотреть список пользователей. Но вместе с тем, стоит учитывать, что доступ к персональным данным пользователей должен быть у строго ограниченного числа лиц. В рамках системы только пользователи с ролью типа «Администратор» имеют доступ к персональным данным других пользователей, который позволяет им устанавливать роли для этих учетных записей, а также просматривать все внутренние данные, которые с ними связаны.

С течением времени в подобного рода сервисах скапливается достаточное количество пользователей, которые не осуществляют работу с системой,

или же которые по разным причинам утратили доступ к своему аккаунту. Для повышения производительности системы и очистки неактуальных данных, администратор может воспользоваться деактивацией старых аккаунтов. При деактивации аккаунта происходит удаление пользовательских данных из системы, однако удаление совершенных заказов и списка просмотренных им товаров затронут не будет. Это реализовано с целью возможности формировать статистические данные.

2.3 Разработка информационной модели

На основании функциональной модели была разработана информационная модель программного решения. На рисунке 2.5 представлена информационная модель проектируемой системы.

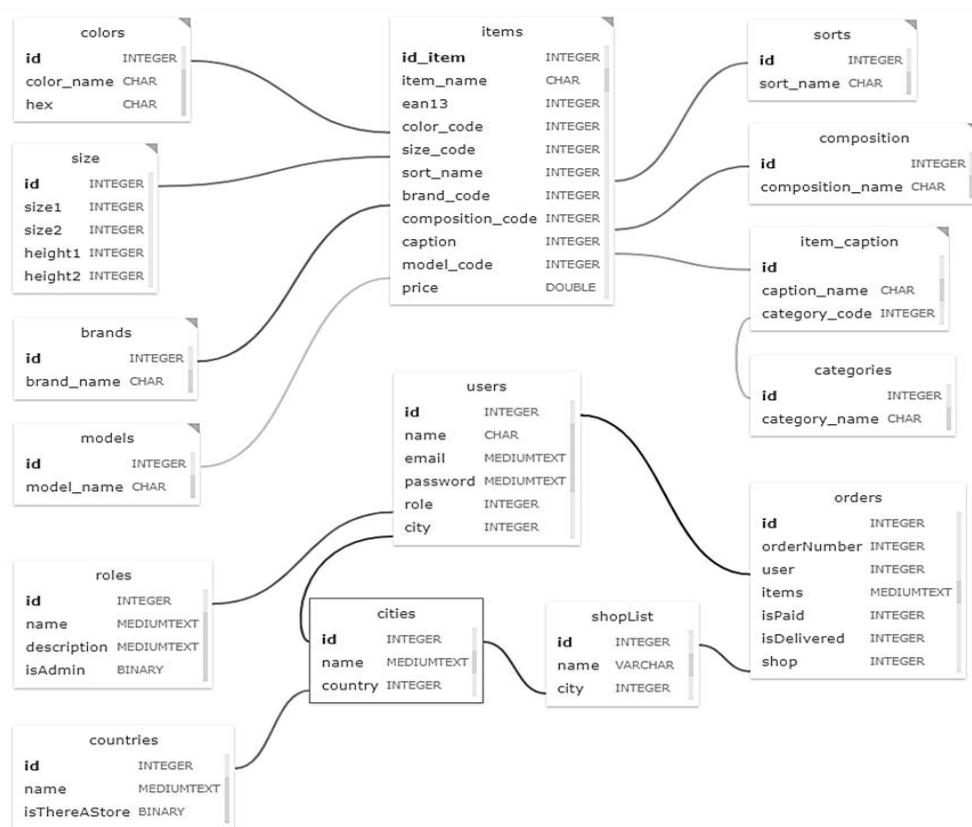


Рисунок 2.5 – Информационная модель проектируемой системы

В процессе анализа предметной области были выделены следующие типы сущностей:

- User – сущность, хранящая представляющая собой список пользователей в рамках системы. Атрибут Id – идентификатор пользователя, Name – имя пользователя, Surname – фамилия пользователя, Email – электронный адрес пользователя, Password – хэш пароля пользователя, PasswortSalt – случайное значение, использованное при хешировании пароля, RoleType – тип роли пользователя, City – город, указанный пользователем при регистрации.

- Role – таблица, хранящая все заданные типы ролей учетных записей. Атрибут Id – идентификатор роли, Name – краткое наименование роли, Description – полное описание роли, IsAdmin – флаг, указывающий на принадлежность данной роли к администраторам.
- Cities – список городов. Атрибут Id – идентификатор города, name – название города, country – страна, isThereAStore – флаг, означающий наличие хотя бы одного магазина в данном городе.
- Country – список стран. Атрибут id – идентификатор, name – название страны, StoreCount – число, означающее количество магазинов в данной стране.
- ShopList – сущность, содержащая список всех действующих магазинов для реализации возможности самовывоза. Атрибут id – идентификатор магазина, name – название магазина, city – номер города, address – адрес магазина.
- Items – одна из главных сущностей, которая содержит в себе всю информацию о имеющихся товарах. Атрибут id_item – идентификатор товара, item_name – наименование товара, ean13 – штрих-код в формате EAN-13, color_code – код цвета, size_code – код размера, sort_name – код сорта товара, brand_code – код торговой марки либо коллекции, composition, caption – полное наименование товара, model_code – номер модели, price – цена на момент получения товара.
 - Colors – список цветовых вариаций моделей. Атрибут id – идентификатор цвета, color_name – наименование цвета, hex – код цвета в формате HEX.
 - Size – сущность, хранящая в себе данные о всевозможно допустимых вариациях размера-роста. Атрибут id – идентификатор размера, size1 – размер 1, size2 – размер 2, height1 – рост 1, height2 – рост 2, fullSize – полное обозначение размера.
 - Brands – список коллекций и торговых марок. Атрибут id – идентификатор коллекции, brand_name – наименование коллекции, isBrand – флаг, отвечающий за обозначение торговой марки или бренда.
 - Models – сущность, представляющая собой сведения о моделях. Атрибут id – идентификатор модели, model_name – номер модели, sizeRange – диапазон размеров модели.
 - Sorts – список всевозможных сортов изделий. Атрибут id – идентификатор сорта, sort_name – обозначение сорта.
 - ItemCaption – сущность, содержимым которой является список наименований изделий. Атрибут id – идентификатор наименования, caption_name – наименование изделия, category_code – код категории, которой будет присвоен товар при добавлении в таблицу Items.
 - Categories – список категорий, на которые будет разделяться весь объем товаров. Атрибут id – идентификатор категории, category_name – наименование категории, level – уровень категории.

- Orders – вторая по важности сущность. В ней содержится вся информация о совершенных пользователями заказах. Атрибут id – идентификатор заказа, orderNumber – внутриорганизационный номер заказа, user – идентификатор пользователя, совершившего заказ, items – список пар «идентификатор товара – количество», отвечающих за определение содержимого корзины пользователя на момент совершения заказа, isPaid – флаг, положение которого меняется в зависимости от статуса оплаты заказа, isDelivered – показатель доставки заказа, shop – идентификатор магазина, в котором будет собран заказ для последующего самовывоза покупателем, orderDate – дата и время совершения заказа.

2.4 Разработка модели взаимодействия пользователя с интерфейсом

В процессе анализа программных решений в области дипломного проектирования, отмечалась необходимость обеспечения высокого уровня интерактивности при взаимодействии с пользователем. На рисунке 2.6 изображена модель взаимодействия пользовательского интерфейса, управляющей логики и данных программного средства, именуемая MVC.

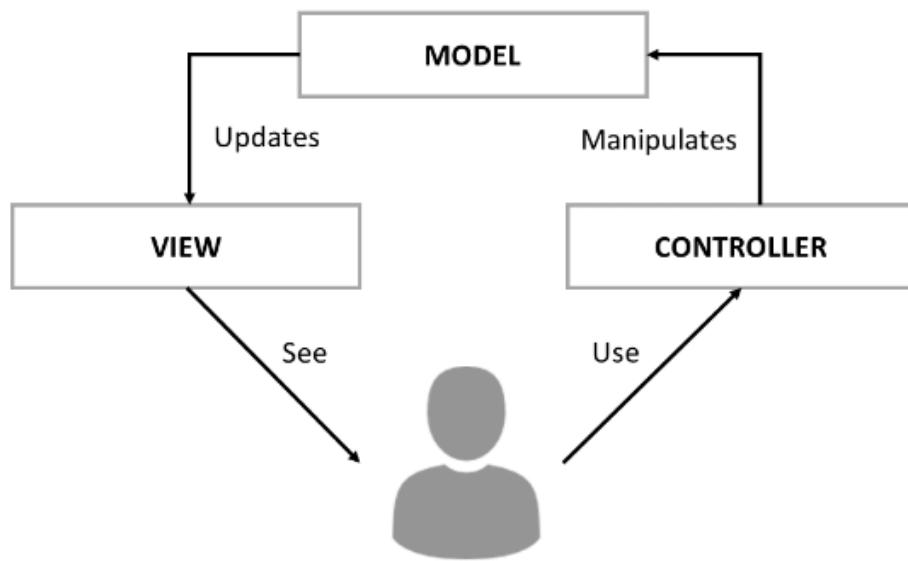


Рисунок 2.6 – Схема MVC

На данной диаграмме отражены классические элементы MVC. Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

На данной модели отображены следующие компоненты:

- модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние;

- представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели;
- контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления, т.е. фактически ей не известно, каким способом визуализировать данные и контроллера, не имеющего точек взаимодействия с пользователем, просто предоставляя доступ к данным и управлению ими. Модель строится таким образом, чтобы отвечать на запросы, изменения своё состояние, при этом может быть встроено уведомление наблюдателей. Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной модели.

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

Контроллер обеспечивает связь между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

2.5 Разработка спецификации функциональных требований

Исходя из результатов анализа исходных данных для проектируемого программного средства, можно выделить следующее: основной целью работы является создание программного продукта, который позволил бы разрешить существующие недостатки программных средств в области, рассматриваемой в процессе дипломного проектирования.

Среди недостатков программных решений стоит отметить следующие:

- наличие недоработок мобильных версий сайта;
- отсутствие детализации предлагаемых продуктов;
- недостаточный уровень удобства пользования программным средством;
- отсутствие оптимизации загрузки данных;
- отсутствие возможности пользования средством с нескольких устройств одновременно.

В ходе проектирования и разработки необходимо продумать и реализовать следующие функциональные возможности:

- управление каталогом;
- наличие категорий и подкатегорий товаров;
- изменение вида каталога продукции;
- сортировка товаров каталога по трем параметрам;
- выбор типа валюты для отображения цен в каталоге;
- глобальный поиск по нескольким параметрам товара;

- выбор диапазона цен и размеров в каталоге;
- выбор одного либо нескольких цветов изделий;
- регистрация и авторизация пользователей;
- перемещение товаров в корзину;
- оформление заказов в режиме онлайн;
- выбор способа доставки и оплаты.

Разрабатываемое решение представляет собой программное средство, предназначенное для предоставления покупателям возможности совершать покупки в режиме онлайн. Основными функциями данного программного средства являются следующие:

- функции идентификации и аутентификации;
- система должна поддерживать создание заказов;
- система должна корректно работать на разных устройствах и операционных системах;
- система должна иметь удобный и интуитивно понятный интерфейс;
- возможность назначения администраторов системы.

Идентификация и аутентификация должны быть выполнены в соответствие со следующими требованиями:

- авторизация должна быть осуществлена по адресу электронной почты и персональному паролю;
- следует предусмотреть возможность аутентификации одного и того же пользователя с нескольких устройств.

2.6 Разработка технических требований к программному продукту

Разрабатываемое программное решение должно обеспечивать корректное функционирование при развертывании программных модулей на сервере со следующими техническими характеристиками:

- процессор – Xeon 2.2 ГГц или более быстродействующий процессор с поддержкой инструкций типа SSE;
- оперативная память – 4 Гбайт или более;
- доступ к сети Интернет;
- доступный объем дискового пространства – не менее 10 Гбайт.

Для нормального функционирования клиентской части программного продукта должны выполняться следующие технические требования:

- Intel Pentium 4/Athlon 64 или более поздней версии;
- свободное место на диске 350 Мб и более;
- оперативная память 2 Гб или более;
- браузеры Microsoft Edge версии 63 и выше, Google Chrome версии 63 и выше, Opera версии 58 и выше, Mozilla Firefox 75 и выше.

Поскольку разработанное программное средство является веб-приложением, то определение требований к операционным системам является нецелесообразным.

3 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка архитектуры программного продукта

Закончив формулирование функциональных требований к разрабатываемой системе, а также исходя из результатов анализа существующих программных средств, можно определить основные моменты организации системы, в рамках которой будет функционировать разрабатываемое программное решение.

Процесс проектирования архитектуры программного обеспечения включает в себя сбор требований клиентов, их анализ и создание проекта в соответствии с требованиями. Удачная разработка программных средств заключается в балансе компромиссов и противоречащих требований, соответствии принципам проектирования и рекомендованным методам, выработанным со временем и дополняется современным оборудованием, сетями и системами управления.

Сопоставив цели определенного компонента программного средства со сведениями о его реализации в коде, мы получим архитектуру программного средства. Правильное понимание архитектуры обеспечит оптимальный баланс требований и результатов. Только программное обеспечение с хорошо продуманной архитектурой способно выполнять указанные задачи с параметрами исходных требований, одновременно обеспечивая максимально высокую производительность.

Программные средства в данной предметной области построены на базе клиент-серверной архитектуры. Использование других подходов приводит к сложностям связанным, со сбором результатов заполнения форм, а также синхронизацией моделей форм между отдельными клиентами внутри системы.

На рисунке 3.1 представлена диаграмма развертывания программного средства. На данной диаграмме нашел свое отражение клиент-серверных подход. Помимо блоков, отражающих рабочие устройства пользователей, на диаграмме представлен сервер, на котором развернуто приложение.

Сервер приложений представляет собой устройство, на котором развернут веб-сервер и серверная часть приложения. В рамках клиент-серверной архитектуры веб-сервер представляет собой посредника между логикой программного средства и внешним миром. Его основная задача: принимать запросы от пользователей и вызывать соответствующие методы обработки данных запросов, а также давать ответ браузеру пользователя, который, в свою очередь, будет отрисовывать страницу пользователю.

Помимо прочего, среди функций, реализуемых веб-сервером, есть и возможность возврата статического контента. Статический контент включает изображения, стили, клиентские модули приложения.

Диаграмма развертывания изображена на рисунке 3.1.

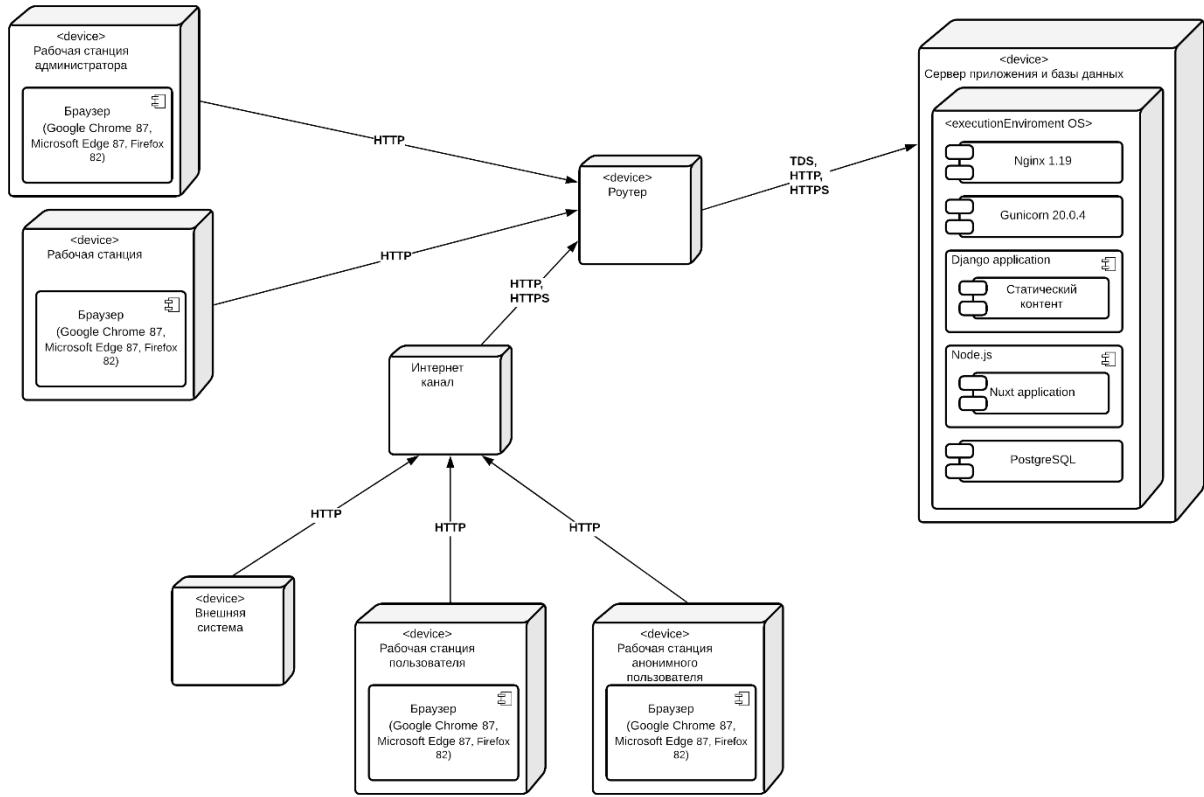


Рисунок 3.1 – Диаграмма развертывания

В качестве хранилища данных необходимых для функционирования системы используется PostgreSQL. Сервер базы данных может быть размещен как на сервере, на котором располагается само приложение, так и на других серверах, доступных в рамках данной сети. Помимо прочего существует возможность использования кластера для обеспечения работы базы данных.

В процессе исследования программных решений в области проведения онлайн-исследований было отмечен факт того, что зачастую клиентская часть приложения реализуется в виде Single Page Application (SPA) [10].

Одностраничное приложение – это веб-приложение или веб-сайт, использующий единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript [9]. Клиентский модуль проектируемого решения должен быть выполнен в соответствии с этой парадигмой.

Все функции, реализованные в результате разработки приложения разбиты на ряд самостоятельных компонентов. Основные архитектурные компоненты программного средства отображены на рисунке 3.2.

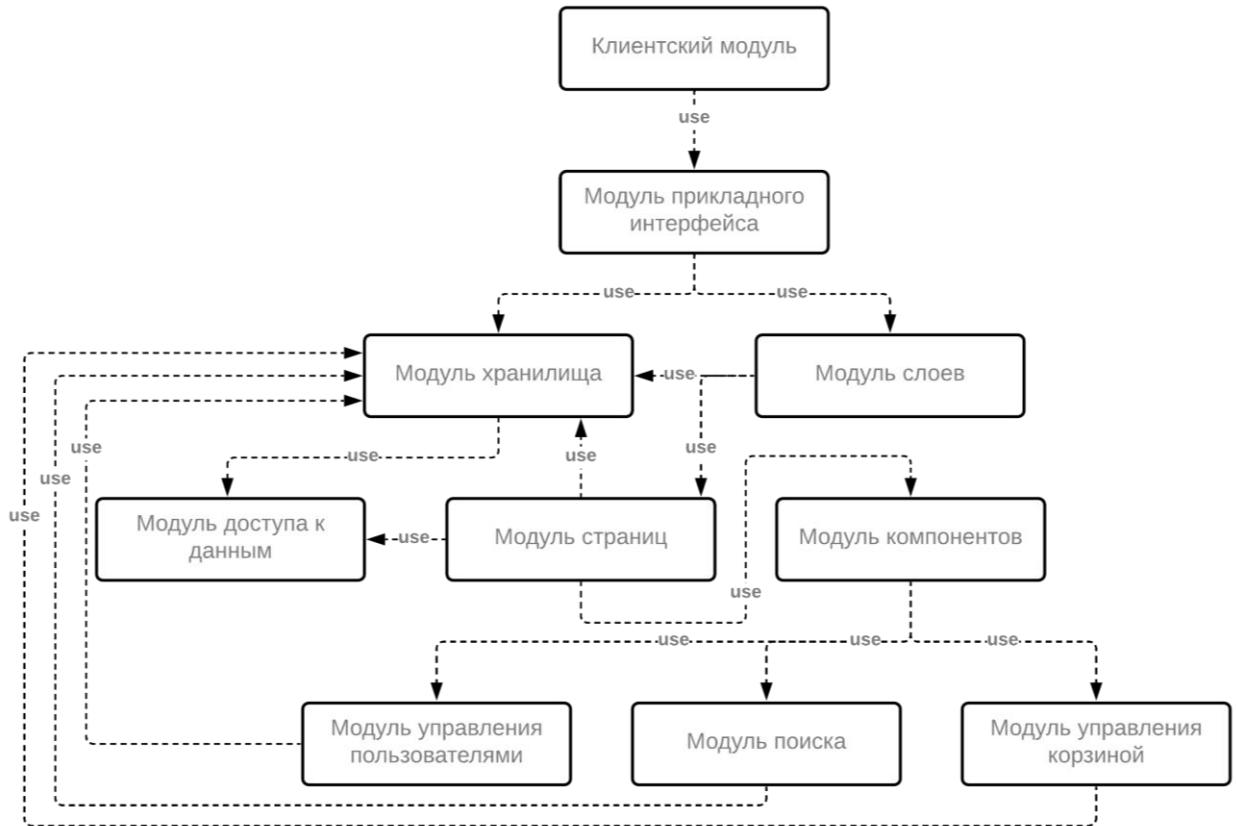


Рисунок 3.2 – Модель архитектуры программного средства

Модулями в данной архитектуре являются отдельные компоненты клиентской части приложения, выполняющие определенную роль.

Модуль слоев. Одна важная вещь, что следует отметить – разделение ответственности – это не то же самое, что разделение на файлы по типу. В современной разработке UI вместо разделения кодовой базы на три огромных слоя, что тесно переплетаются друг с другом, имеет больше смысла делить их на слабо связанные компоненты и компоновать уже их. Внутри компонента, его шаблон, логика и стили неразрывно связаны между собой, что позволяет сделать компонент более сплочённым и удобным в поддержке.

Модуль хранилища. В центре любого Vuejs-приложения находится хранилище. Хранилище – это контейнер, в котором хранится состояние вашего приложения. Два момента отличают хранилище Vue от простого глобального объекта: хранилище Vue реактивно. Когда компоненты Vue полагаются на его состояние, то они будут реактивно и эффективно обновляться, если состояние хранилища изменяется; нельзя напрямую изменять состояние хранилища. Единственный способ внести изменения – явно вызвать мутацию. Это гарантирует, что любое изменение состояния оставляет след и позволяет использовать инструментарий, чтобы лучше понимать ход работы приложения.

Модуль компонентов. Компоненты – это переиспользуемые экземпляры Vue со своим именем.

На рисунке 3.3 представлена схема работы программы в режиме работы администратора. На данной диаграмме наиболее полно отражены функции, реализуемые программным средством.

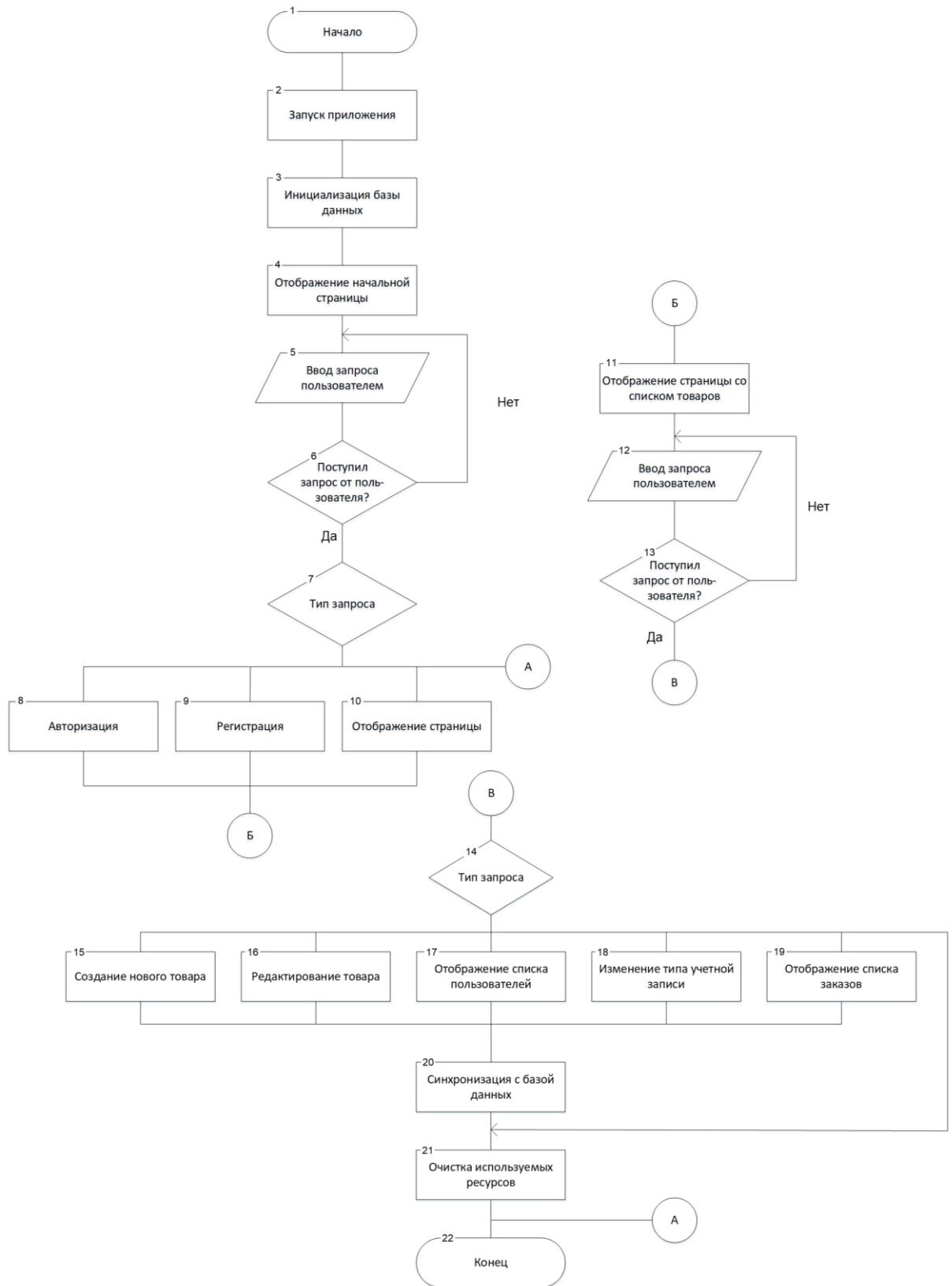


Рисунок 3.3 – Схема работы ПС в режиме администратора

3.2 Проектирование алгоритмов ПС

В процессе проектирования архитектуры был определен ряд алгоритмов реализации логики функций приложения. Более детально были исследованы функции реализации условной логики для каталога товаров, такие как алгоритм применения фильтров и алгоритм поиска товаров, а также алгоритм регистрации пользователей на сайте.

3.3.1 Алгоритм применения фильтров

Одной из функциональных возможностей проектируемого приложения является возможность задания правил отображения товаров на странице. Схема алгоритма, применяемого при реализации данной функциональности, приведена на рисунке 3.4.

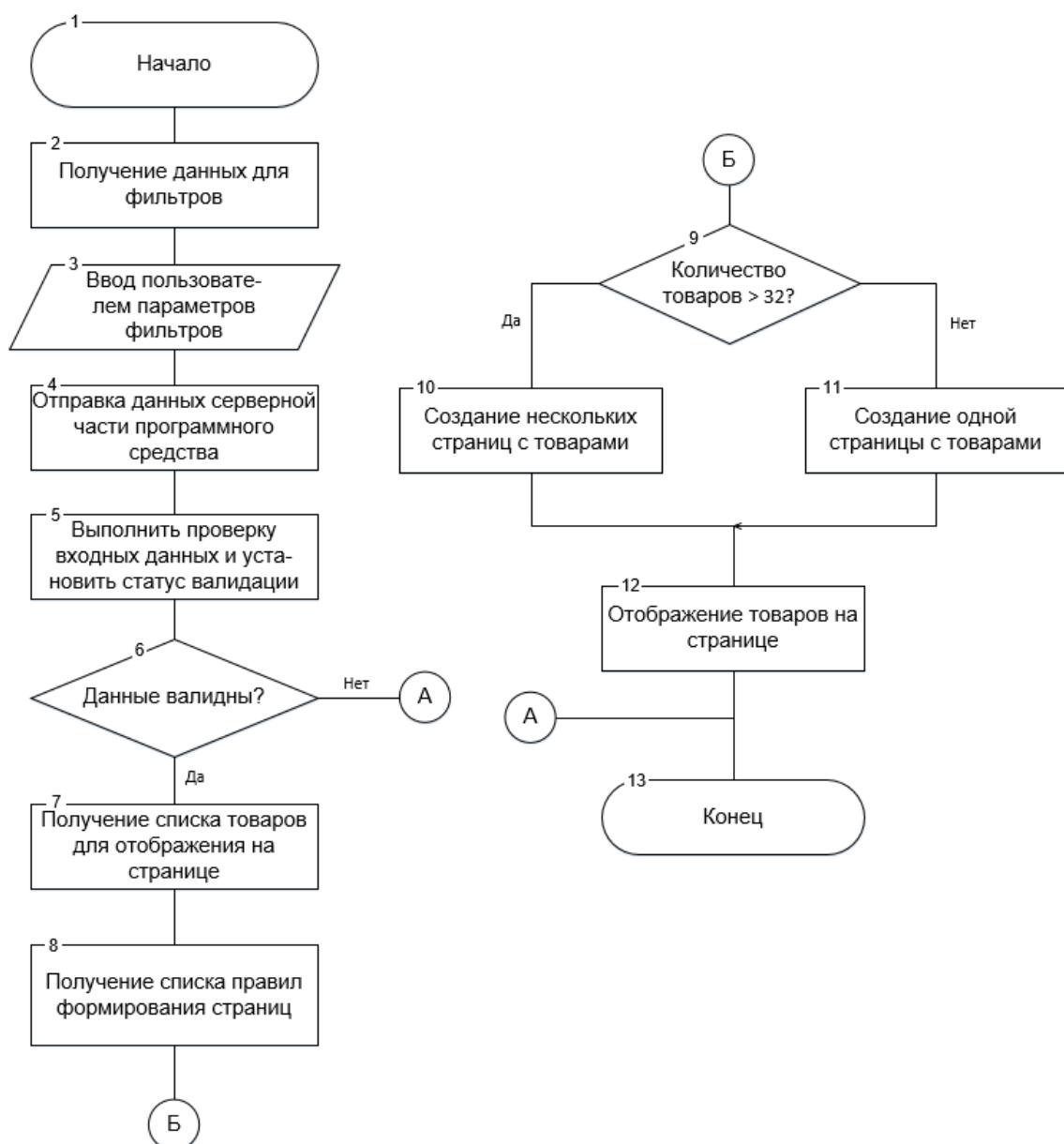


Рисунок 3.4 – Схема алгоритма применения фильтров

Как мы видим из схемы алгоритма, логика работы клиентского модуля основывается на принципах реактивного программирования. После того, как пользователь задаст необходимые ему данные в фильтрах и выполнения заданных проверок корректности введенных данных, выполняется соответствующее действие из вспомогательного класса.

В свою очередь действия вызывают диспетчер, что приводит к обновлению уже отрисованных компонентов. Списковая компонента запрашивает правила отображения для каждого элемента на странице, после чего пробегается по полученному списку правил в поисках такого, условие которого выполняется.

В случае, если найдено более одного правила, система должна должна отобразить каждое найденное правило. Таким образом мы имеем возможность однозначно определить необходимое поведение системы.

В зависимости от типа правила отображения, делается вывод о необходимости отображение данного товара на странице.

В процессе реализации данного алгоритма следует использовать функциональный подход. Использование чистых функций положительно сказывается на реактивных интерфейсах, так как сам подход предполагает отсутствие состояний.

3.3.2 Алгоритм поиска товаров

Еще одной возможностью, схожей с логикой правил отображения является логика поиска товаров, удовлетворяющих критериям пользователя. Поле поиска для сайта — один из важнейших элементов пользовательского интерфейса на веб-странице. С его помощью пользователь может найти необходимый ему контент на сайте, указав либо полное наименование товара или номера модели, либо частичное. Поиск срабатывает при вводе трех и более символов в строку поиска. Принципиальное отличие в данном случае заключается в том, что критерием выбора результата является один из двух параметров: наименование модели и ее номер.

Как и в случае с фильтрами, при условии, что критерий отображения выполнен в более чем одном правиле, будет отображено несколько товаров, правила выбора которых были выполнены, в противном случае пользователь будет уведомлен об отсутствии товаров, подходящих под условия поиска. Алгоритм поиска товаров приведен на диаграмме 3.5.

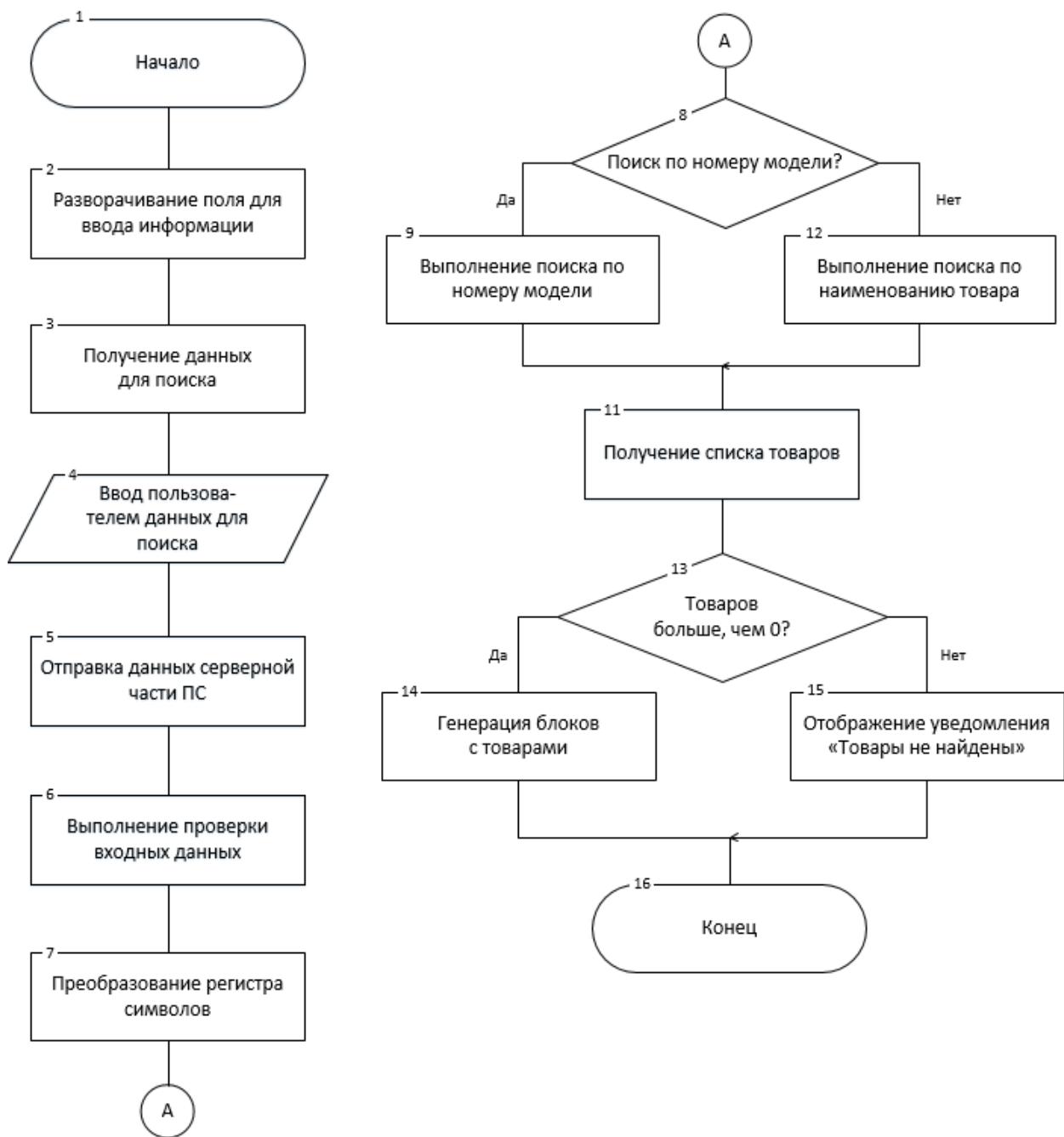


Рисунок 3.5 – Схема алгоритма поиска товаров

В ситуации, когда не был выполнен ни один критерий, вместо генерации блоков с товарами пользователю будет отображаться уведомление, гласящее о нулевом результате поиска товаров.

4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

На основе спецификации функциональных требований и спроектированной архитектуры программного средства, а также требований к техническим характеристикам аппаратного обеспечения был произведен выбор подходящих технологий для разработки программного решения.

4.1 Выбор и обоснование языков программирования

Программное средство можно разделить две основные части: серверная часть, обеспечивающая работу с базой данных и реализующая сохранение результатов работы с программным средством, и клиентская часть, реализующая интерфейс для работы с программным средством.

4.1.1 Серверная часть ПС

Серверная часть программного средства реализована с использованием языка программирования Python и фреймворка для веб-приложений Django.

Django – свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC [4].

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Это одно из существенных архитектурных отличий этого фреймворка от некоторых других. Один из основных принципов фреймворка – не повторяться.

Также, в отличие от других фреймворков, обработчики URL в Django конфигурируются явно при помощи регулярных выражений.

Для работы с базой данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

Архитектура Django похожа на «Модель-Представление-Контроллер» (MVC). Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представление (View), а презентационная логика Представления реализуется в Django уровне Шаблонов (Template). Из-за этого уровневую архитектуру Django часто называют «Модель-Шаблон-Представление» (MTV).

Первоначальная разработка Django как средства для работы новостных ресурсов достаточно сильно отразилась на его архитектуре: он предоставляет ряд средств, которые помогают в быстрой разработке веб-сайтов информационного характера. Так, например, разработчику не требуется создавать контроллеры и страницы для административной части сайта, в Django есть встроенное приложение для управления содержимым, которое можно включить в любой сайт, сделанный на Django, и которое может управлять сразу несколькими сайтами на одном сервере. Административное приложение позволяет создавать, изменять и удалять любые объекты наполнения сайта, протоколируя

все совершенные действия, и предоставляет интерфейс для управления пользователями и группами (с пообъектным назначением прав).

Некоторые возможности Django:

- ORM, API доступа к БД с поддержкой транзакций;
- встроенный интерфейс администратора, с уже имеющимися переводами на многие языки;
- диспетчер URL на основе регулярных выражений;
- расширяемая система шаблонов с тегами и наследованием;
- система кеширования;
- интернационализация;
- подключаемая архитектура приложений, которые можно устанавливать на любые Django-сайты;
- шаблоны функций контроллеров;
- авторизация и аутентификация, подключение внешних модулей аутентификации: LDAP, OpenID и другие;
- система фильтров для построения дополнительных обработчиков запросов, как например включённые в дистрибутив фильтры для кеширования, сжатия, нормализации URL и поддержки анонимных сессий;
- библиотека для работы с формами (наследование, построение форм по существующей модели БД);
- встроенная автоматическая документация по тегам шаблонов и моделям данных, доступная через административное приложение.

4.1.2 Клиентская часть ПС

В качестве языка и фреймворка, на которых велась разработка клиентского модуля приложения были выбраны язык программирования JavaScript и Nuxt.js.

Nuxt.js – это фреймворк, позволяющий создавать универсальные веб-приложения на Vue.js с использованием Node.js. С помощью него можнорендерить UI на сервере и генерировать статические сайты.

Основные принципы и возможности фреймворка Nuxt:

- Серверные страницы предоставляют полную HTML-страницу, уже готовую для рендера, а затем страница «гидрируется» с помощью JavaScript в клиенте для добавления интерактивности и превращается в одностраничное приложение (SPA). Это означает, что приложения Nuxt представляют собой отдельные приложения JavaScript, которые загружают данные с использованием интерфейса API, а не внедряются в другие серверные фреймворки приложений или накладываются на статический HTML.
- Предварительная загрузка асинхронных данных. Чтобы реализовать на сервере преимущества предварительно отображаемых страниц, вам необходимо обеспечить, чтобы движок рендера на сервере имел все необходимые данные перед началом рендера. Это просто для статических страниц, но для динамических приложений, зависящих от вызовов API, вам необходимо

обеспечить, чтобы все критические данные извлекались до того, как страница будет отображена и отправлена с сервера. Даже для чистых SPA-приложений полезно иметь хуки, которые позволяют указать, какие данные необходимы, прежде чем страница будет визуализирована, а что может быть заполнено позже.

– Разбор страниц на макеты, страницы и компоненты. Одна из лучших особенностей архитектуры на основе компонентов заключается в том, что все можно рассматривать как компонент. Однако при переходе в систему, использующую для создания отдельных страниц маршрутизацию, полезно добавить еще некоторую структуру поверх этой абстракции. Nuxt позволяет делать это, используя понятия страниц и макетов. Страница соответствует маршруту и, естественно, тому, как мы привыкли думать о сети. Каждая страница может иметь макет, который отображается внутри нее, поэтому макеты становятся способом создания общей структуры на разных страницах.

– Организация структуры файлов по ролям. Один из первых вопросов в любом виде приложений – как организовать файлы. Nuxt использует относительно простой подход, сохраняя при этом структуру. Файлы разделены в соответствии с ролями, с каталогами для components, layouts, pages, middleware, plugins, store, компилируемых assets и полностью static ресурсов.

4.2 Выбор среды разработки

В качестве среды разработки для написания исходного кода программного средства была выбрана Visual Studio Code.

Visual Studio Code – редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для настройки: пользовательские темы, сочетания клавиш и файлы конфигурации. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом, но готовые сборки распространяются под проприetaryной лицензией.

Палитра команд представляет собой подобие командной строки, которая вызывается сочетанием клавиш.

Visual Studio также позволяет заменять кодовую страницу при сохранении документа, символы перевода строки и язык программирования текущего документа.

С 2018 года появилось расширение Python для Visual Studio Code с открытым исходным кодом, которое предоставляет разработчикам широкие возможности для редактирования, отладки и тестирования кода.

4.3 Диаграммы классов программного средства

Для наглядной демонстрации всех классов, разработанных в процессе работы над серверной частью программного средства, был построен ряд диаграмм классов, которые, в свою очередь подразделены на определенные модули, каждый из которых отвечает за логику описываемого компонента программного средства.

Для удобства описания диаграммы классов ее можно подразделить на три условные части: классы логики работы с товарами, логику разделения ролей и авторизации, а также классы, реализующие работу с сессиями.

Первая рассматриваемая часть – модули product и reference, отвечающие за логику работы с товарами. Классы и связи данного модуля изображены на рисунке 4.1.

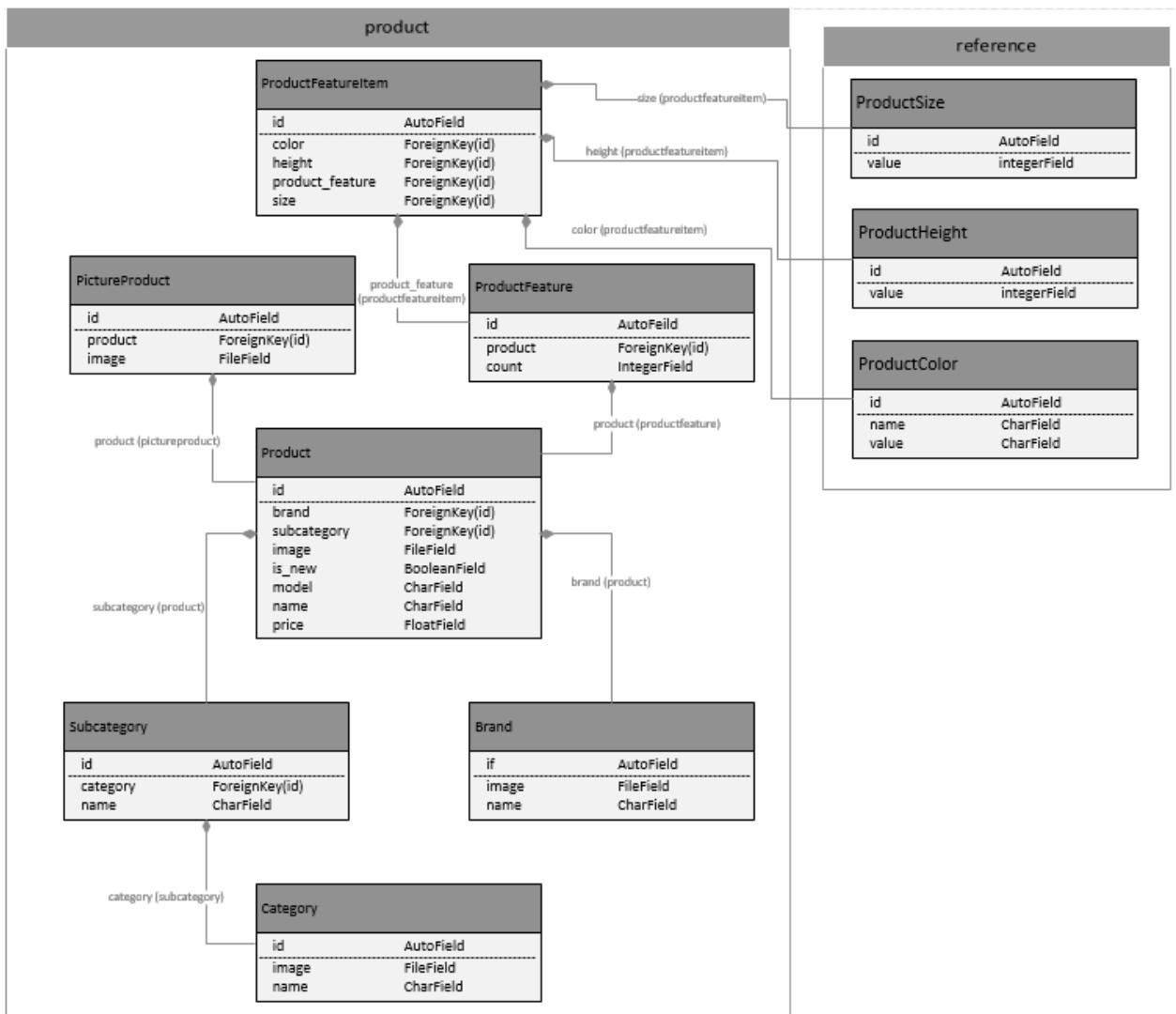


Рисунок 4.1 – Диаграмма классов модуля работы с товарами

На данной модели перечислены все классы, которые используются для реализации логики работы с товарами. Большая часть классов представляет

собой типы данных описывающих модели предметной области, а также перечисления, которые используются, для реализации логики в предметной области приложения.

Основным классом данного модуля является `product`, который хранит информацию о товарах, отображаемых в каталоге. Вспомогательными классами, описывающими детализацию продуктов являются `subcategory` и `category`, определяющие категорию и подкатегорию заданного товара, `brand`, отвечающий за торговую марку товара, `PictureProduct`, - изображения товара, `ProductFeature` – количество имеющихся единиц товара.

Следующей рассматриваемой частью являются модули администрирования и авторизации (рисунок 4.2).

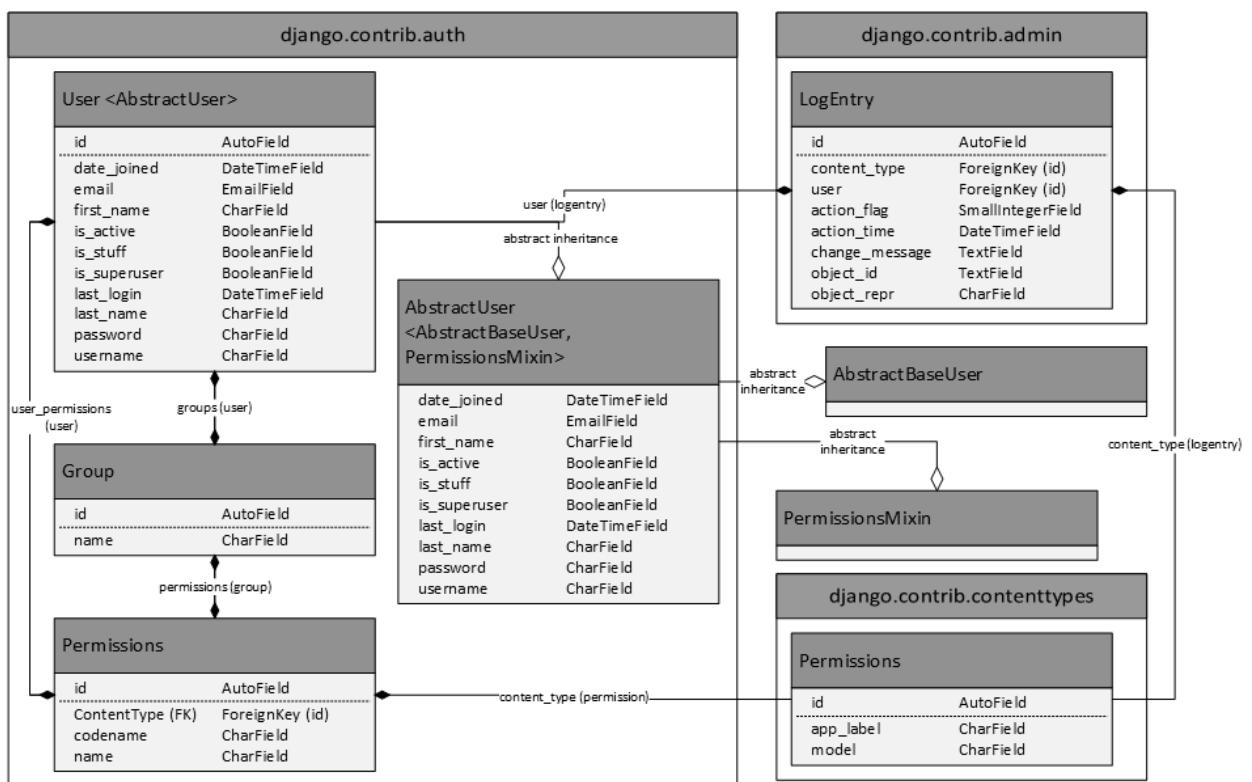


Рисунок 4.2 – Диаграмма классов авторизации и администрирования

На рисунке изображены классы, реализующие авторизацию пользователя в системе, а также разделения ролей между пользователями. Логика авторизации реализована с использованием паттерна посредник. Данный подход позволяет использовать разработанную компоненту при реализации аналогичного модуля в любом другом приложении. Изображенные на диаграмме классы осуществляют проверку корректности ключевого значения, а также генерацию токена при входе пользователя в систему.

Заключающим модулем является группа классов, реализующая работу приложения с сессиями пользователей. Данная группа классов изображена на рисунке 4.3.

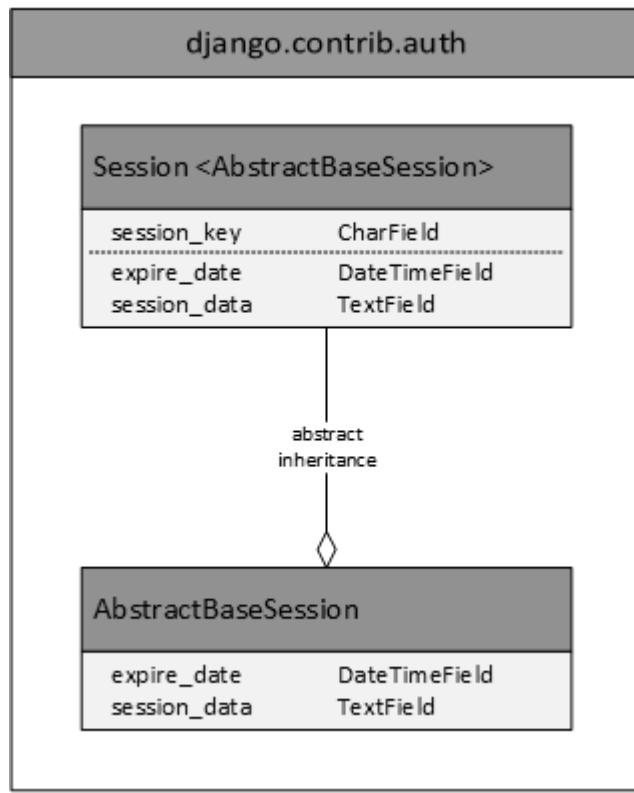


Рисунок 4.3 – Диаграмма классов логики сессий

Изображенный на диаграмме класс осуществляет логику работы программного средства с сессиями пользователей. Django полностью поддерживает сессии для анонимных пользователей, позволяет сохранять и получать данные для каждого посетителя сайта. Механизм сессии сохраняет данные на сервере и самостоятельно управляет сессионными cookies.

4.4 Описание компонентов клиентской части программного продукта

Для реализации клиентской части на Nuxt все элементы управления были разбиты на отдельные компоненты (модули). Каждая отдельная компонента предназначена для реализации отображения одного атомарного элемента.

Некоторые разработанные компоненты представляют собой сложные и используют другие. Такие высокоуровневые компоненты реализуют целые страницы или формы приложения.

Все разработанные компоненты можно отнести к одной из следующих групп: компоненты страниц, компоненты хранилища, компоненты слоев и вспомогательные компоненты.

Каждая описанная компонента содержит ряд методов, описывающих ее работу. Среди них можно выделить следующие: `getInitialState()`, `ComponentDidMount()`, `componentWillMount()`, `ComponentWillUnmount()`, `render()`, `getProps()`.

Метод `getInitialState()` содержит программный код, который осуществляет инициализацию состояния компоненты перед первым отображением компоненты на странице. Данный метод вызывается один раз и позволяет задать начальные условия отображения, или же осуществить подготовку данных.

Метод `componentWillMount()` также вызывается один раз перед внедрением компоненты в дерево узлов на веб-странице. В данном методе как правило осуществляется подписка на события, по которым будет осуществляться дальнейшая перерисовка компоненты. Как правило в данном методе осуществляется подписка на обновление значений в хранилище данных.

Единственным обязательным для каждой компоненты методом является метод `render()`. В данном методе осуществляется генерация кода HTML, который будет отображаться в месте подключения генерируемой компоненты. Внутри данного метода могут использоваться как другие пользовательские компоненты, так и стандартные для Nuxt компоненты.

Передача параметров между компонентами осуществляется через использование свойств. Согласно лучшим практикам написания кода, данные свойства должны быть описаны в методе `getProps()`. Данный метод возвращает список свойств, поддерживаемых данным компонентом.

5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

5.1 Методы тестирования

Тестирование программного обеспечения – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом [15].

Тестирование – важный этап разработки любого программного обеспечения. В ходе этого процесса проверяется соответствие реального состояния ПС с ожидаемым (или установленным), основываясь на наборе тест-кейсов.

Можно определить такие основные цели тестирования программного обеспечения:

- предоставление информации о качестве ПО конечному заказчику;
- повышение качества ПО;
- предотвращение появления дефектов.

Цели тестирования могут отличаться, в зависимости от этапа разработки ПО, на котором оно проводится. К примеру, на этапе кодирования целью тестирования будет вызов как можно большего количества сбоев в работе программы, что позволит локализовать и исправить дефекты. В то же время, при приемочном тестировании необходимо показать, что система работает правильно. В период сопровождения, тестирование в основном необходимо для того, чтобы убедиться в отсутствии новых багов, появившихся во время внесения изменений.

Модульное тестирование, иногда блочное тестирование или юнит-тестирование – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

Интеграционное тестирование – одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию.

Системное тестирование – это уровень тестирования, который проверяет законченный и полностью интегрированный программный продукт. Целью системного теста является оценка сквозных технических характеристик системы.

Функциональное тестирование является одним из ключевых видов тестирования, задача которого – установить соответствие разработанного программного обеспечения (ПО) исходным функциональным требованиям компании клиента. То есть проведение функционального тестирования позволяет проверить способность информационной системы в определенных условиях решать задачи, нужные пользователям.

5.2 Тест-кейсы

Функциональное тестирование является основным видом тестирования программного обеспечения. Каждая функция программы тестируется и при этом делается вывод об ее правильности. Очевидно, что по всей области определения проверить функцию невозможно и поэтому каждая функция проверяется на правильность в некоторых точках области её определения.

Данный метод тестирования позволяет:

- обнаружить некорректные или отсутствующие функции;
- обнаружить ошибки интерфейса;
- обнаружить ошибки во внешних структурах данных (файлы, базы данных).

В рамках разработки дипломного проекта функциональное тестирование реализовано с помощью набора тест-кейсов, собранного в тестовый сценарий, представленный в таблице 5.1.

Таблица 5.1 – Тестирование интерфейса и функциональности программного средства

№	Действие	Ожидаемый результат	Результат теста
1	2	3	4
1	1) Открыть приложение. 2) Нажать на кнопку «Профиль».	Отображение формы авторизации пользователя.	Совпадает с ожидаемым.
2	1) Открыть приложение. 2) Нажать кнопку «Профиль». 3) Нажать кнопку «Вход». 4) Ввести данные не зарегистрированного пользователя. 5) Нажать кнопку «Войти»	Получение сообщения о неправильно введенных данных.	Совпадает с ожидаемым.
3	1) Открыть приложение. 2) Нажать кнопку «Профиль». 3) Нажать кнопку «Вход». 4) Ввести данные учетной записи администратора. 5) Нажать кнопку «Войти».	Пользователь успешно авторизован, открыта главная страница программного средства.	Совпадает с ожидаемым.
4	1) Открыть приложение. 2) Нажать кнопку «Профиль». 3) Нажать кнопку «Регистрация». 4) Оставить обязательные для заполнения поля пустыми. 5) Нажать кнопку «Регистрация».	Вывод ошибки, указывающей на пустое обязательное к заполнению поле.	Совпадает с ожидаемым.

Продолжение таблицы 5.1.

1	2	3	4
5	1) Открыть приложение. 2) Нажать кнопку «Профиль». 3) Нажать кнопку «Регистрация». 4) Ввести данные в соответствующие им поля. 5) Нажать кнопку «Регистрация».	Вывод сообщения об успешной регистрации пользователя.	Вывод сообщения об успешной регистрации пользователя.
6	1) Открыть приложение. 2) Ввести данные учетной записи пользователя. 3) Нажать кнопку «Войти». 4) Нажать кнопку «Выйти».	Переход на главную страницу.	Совпадает с ожидаемым
7	1) Открыть приложение. 2) Ввести данные учетной записи администратора. 3) Нажать кнопку «Войти». 4) Нажать кнопку «Выйти».	Переход в защищенную область программного средства.	Совпадает с ожидаемым
8	Открыть приложение повторно после авторизации в системе, в течение 5 минут с момента первой авторизации.	Пользователь по-прежнему авторизован в системе	Совпадает с ожидаемым.
9	Перейти в защищенную область приложения по прямому адресу, не будучи авторизованным в качестве администратора.	Переход на страницу No permission	Совпадает с ожидаемым.
10	Перейти в защищенную область приложения по прямому адресу, будучи авторизованным в качестве администратора.	Переход в защищенную область программного средства.	Совпадает с ожидаемым.
11	1) Перейти на страницу заказа по прямой ссылке. 2) В адресе указать идентификатор еще не созданного заказа.	Переход на страницу Object not found	Совпадает с ожидаемым.
12	Перейти на страницу оформления заказа по прямой ссылке, не имея товаров в корзине.	Переход на страницу Object not found	Совпадает с ожидаемым.
13	Перейти на страницу личного кабинета по прямой ссылке, не авторизовавшись.	Переход на страницу Object not found	Совпадает с ожидаемым.

Продолжение таблицы 5.1.

1	2	3	4
14	1) Открыть каталог товаров. 2) Написать текст, не относящийся к товарам в строку поиска.	Отображение пустой страницы	Совпадает с ожидаемым.
15	1) Открыть каталог товаров. 2) Установить в фильтрах несовместимые параметры товаров.	Отображение надписи «По вашим критериям ничего не найдено»	Совпадает с ожидаемым.
16	1) Открыть каталог товаров. 2) Добавить в корзину понравившийся товар. 3) Открыть корзину.	Отображение товара в корзине	Совпадает с ожидаемым.
17	1) Открыть каталог товаров. 2) Добавить товар в корзину. 3) Нажать кнопку «Перейти к оформлению заказа».	Отображение страницы оформления заказа	Совпадает с ожидаемым.
18	1) Добавить товары в корзину. 2) Нажать кнопку «Перейти к оформлению заказа». 3) На странице оформления заказа ввести все необходимые данные. 4) Нажать кнопку «Оформить заказ».	Отображение уведомления «Заказ успешно оформлен»	Совпадает с ожидаемым.
19	Перейти на страницу оформления заказа по прямому адресу.	Отображения пустой таблицы добавленных в корзину товаров. Кнопка «Оформить заказ» недоступна для нажатия.	Совпадает с ожидаемым.

В результате итогового тестирования серьезных дефектов, влияющих на работу программного средства выявлено не было. Все тестовые случаи были пройдены успешно. Фактический результат совпал с ожидаемым результатом в каждом из тестовых случаев, однако в процессе тестирования было выявлено несколько незначительных ошибок отображения элементов на странице.

5.3 Результаты тестирования

В результате тестирования были обнаружены следующие ошибки:

- некорректное отображение блоков с товарами в корзине на устройствах с шириной экрана меньшей, чем 600 пикселей;

– некорректное отображение контента на странице оформления заказа.

Для устранения найденных ошибок были созданы проверки на действие пользователя и на корректный выбор данных.

Ошибка отображения блоков с товарами в корзине изображена на рисунке 5.1. Блоки товаров, а также блок с итоговой стоимостью товаров в корзине вплотную размещены к левой стороне корзины. При написании стилей для устройств с различным разрешением экрана и соотношением сторон было упущено определение отступа, позволяющего позиционировать товары по центру блока предпросмотра корзины.

	Комплект жен. (фуфайка + брюки) 5 шт.	305 р.	X
	Комплект жен.(жакет + брюки + майка) 18 шт.	280 р.	X
	Комплект жен.(майка + шорты) 318 шт.	105 р.	X
	Комплект жен. (фуфайка + брюки) 1 шт.	41 783 р.	X
	Комплект жен. (фуфайка + брюки) 17 шт.	18 р.	X
	Комплект жен. (фуфайка + брюки) 9999 шт.	7 801 р.	X
	Комплект жен. (фуфайка + брюки) 5 шт.	305 р.	X
	Комплект жен.(жакет + брюки + майка) 18 шт.	280 р.	X
ИТОГО:		1 250 РУБ.	
ПЕРЕЙТИ В КОРЗИНУ			

Рисунок 5.1 – Ошибка отображения блоков с товарами в корзине

Для исправления данной ошибки необходимо скорректировать стили, отвечающие за позиционирование объектов на странице для определенного результата медиа-запроса.

Некорректное отображение контента на странице оформления заказа отображено на рисунке 5.2. Главный, задающий ширину страницы контейнер определил неверную ширину экрана.

The screenshot shows a web application interface for placing an order. At the top, there's a navigation bar with a logo, language selection ('EN'), and search functions. Below the header is a table for selecting items from a catalog. The table columns include 'НАИМЕНОВАНИЕ' (Name), 'МОДЕЛЬ' (Model), 'РАЗМЕР' (Size), 'РОСТ' (Height), 'КОЛ-ВО УП.' (Quantity per unit), 'В УП.' (In unit), 'ВСЕГО' (Total), 'ЦЕНА' (Price), and 'СТОИМОСТЬ' (Cost). A summary row at the bottom of the table shows 'итого:' (Total), '2 УП.' (2 units), '4 ШТ.' (4 pieces), and '6810 РУБ.' (6810 rubles). Below the table, sections for 'Личные данные' (Personal data) and 'Комментарий к заказу' (Comment to order) are visible, along with a text input field for comments.

Рисунок 5.2 – Некорректное отображение контента на странице оформления заказа

Для исправления данной ошибки была произведена коррекция CSS кода.

Результатом описанных выше этапов тестирования являются ошибки. Каждой ошибке был присвоен свой уровень важности, в зависимости от влияния на работу приложения. Ошибки с более высоким приоритетом были исправлены в первую очередь.

Результаты тестирования приложения, а также статистика найденных и исправленных ошибок представлена в таблице 5.2.

Таблица 5.2 – Результаты тестирования

	Критических	Важных	Средних	Незначительных	Всего
1	2	3	4	5	6
Найдено ошибок	0	0	1	1	2
Исправлено ошибок	0	0	1	1	2

По результатам тестирования можно сделать вывод, что полученные характеристики разработанного программного средства приемлемыми, а созданное программное средство работает корректно. Все выявленные ошибки были проанализированы и успешно устранены.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для корректной работы программного средства предъявляются следующие минимальные требования к программным средствам сервера:

- операционная система: Linux (Ubuntu 18);
- веб-сервер и прокси сервер nginx;
- платформа Node.js версии 13 и выше;
- процессор Xeon 2.2 ГГц или более быстродействующий;
- оперативная память 4 Гбайт или более;
- сетевая карта Ethernet 1 Гбит.
- доступный объем дискового пространства 10 Гбайт.м

Дополнительно необходимо наличие развернутой системы управления базами данных Postgres, HTTP-сервер интерфейса шлюза веб-сервера Python – Gunicorn, пакетные менеджер npm.

6.1 Развёртывание сервера программного средства

Готовые модули приложения могут быть поставлены для развертывания в после выполнения следующих требований:

- получение привилегий sudo для нового экземпляра сервера Ubuntu 18.04 с базовым брандмауэром;
- установка следующих пакетов Python:
 1. Python3-pip.
 2. Python3-dev.
 3. Libpq-dev.
 4. Postgresql.
 5. Postgresql-contrib.
 6. Nginx.
- создание виртуальной среды Python для проекта;
- создание базы данных и пользователя PostgreSQL;
- создание и настройка проекта Django;
- создание файлов сокета и служебных файлов systemd для Gunicorn;
- настройка Nginx как прокси для Gunicorn;
- установка пакета pm2 на Ubuntu

Перед началом работы с приложением рекомендуется изменить параметры учетной записи суперпользователя, которая будет создана в процессе развертывания приложения.

Так как приложение для своей работы требует развернутую СУБД, необходимо указать имя базы данных и URL для установления соединения с сервером базы данных. Установка этих параметров производится в файле конфигурирования приложения appsettings.json.

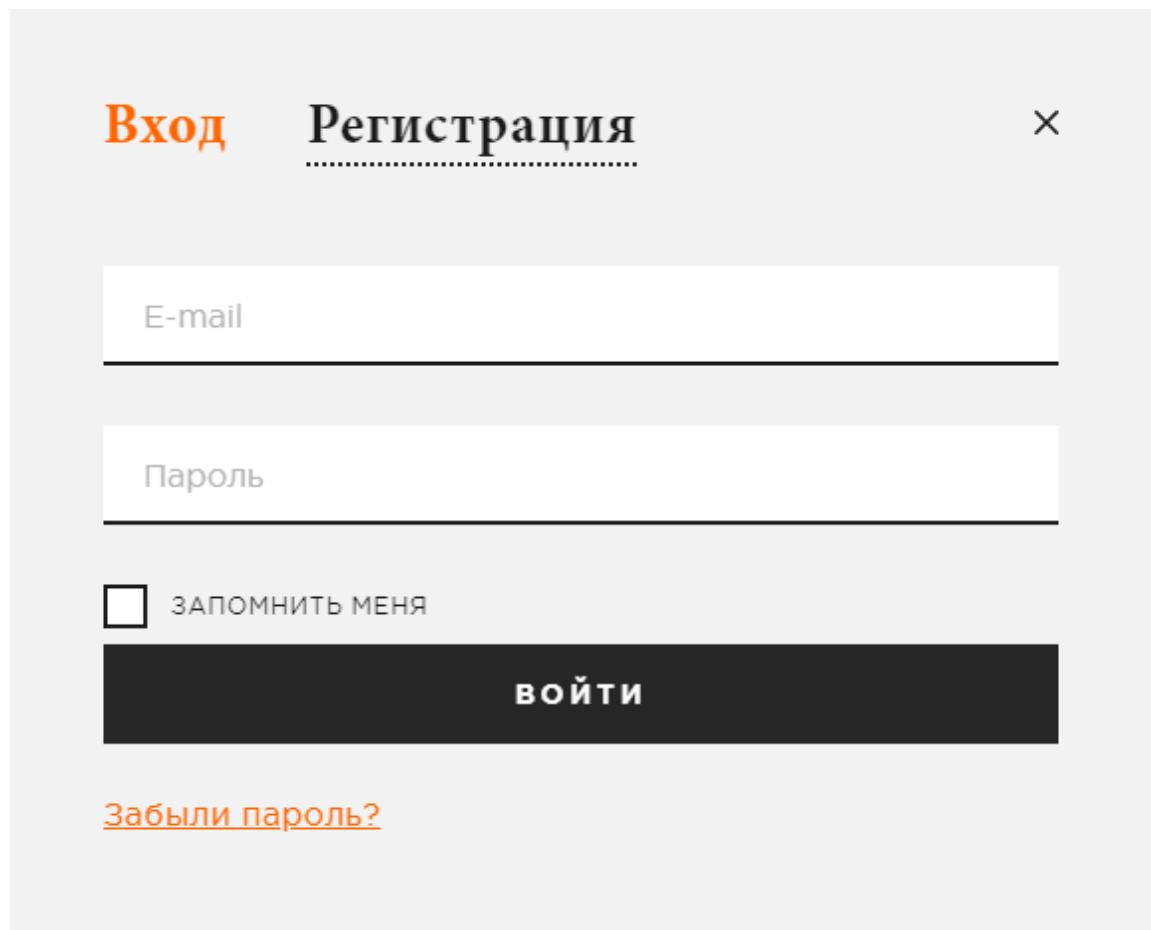
После развертывания приложение по умолчанию будет доступно по адресу 127.0.0.1. Одновременно с запуском веб-сервера будет производится старт приложения.

Клиентский доступ к приложению может быть осуществлен при помощи веб-браузера как персональных компьютеров, так и мобильных устройств.

6.2 Использование программного продукта

6.2.1 Авторизация пользователя

При обращении к системе пользователя не имеющего аккаунта в системе или не прошедшего авторизацию, ему будет отображена страница входа в систему, пример которой представлен на рисунке 6.1.



Форма авторизации пользователя в системе. Видны следующие элементы:

- Логотипы: "Вход" (оранжевый) и "Регистрация" (черный).
- Поля ввода:
 - E-mail (введенная строка: "test@mail.ru")
 - Пароль (введенная строка: "1234567890")
- Переключатель "ЗАПОМНИТЬ МЕНЯ" (не выбран).
- Кнопка "войти" (черная кнопка с белым текстом).
- Ссылка "Забыли пароль?" (оранжевый текст).

Рисунок 6.1 – Форма авторизации пользователя в системе

Для входа пользователя в систему необходимо указать адрес электронной почты и пароль, введенные при регистрации. Задав переключателю «Запомнить меня» активное положение, процесс выхода из учетной записи будет осуществлен спустя значительное количество времени.

6.2.2 Регистрация в системе

В случае, если новый пользователь, не имеющий своей учетной записи, желает создать собственный аккаунт, ему необходимо зарегистрироваться в системе, нажав кнопку «Регистрация». Окно регистрации нового пользователя представлено на рисунке 6.2.

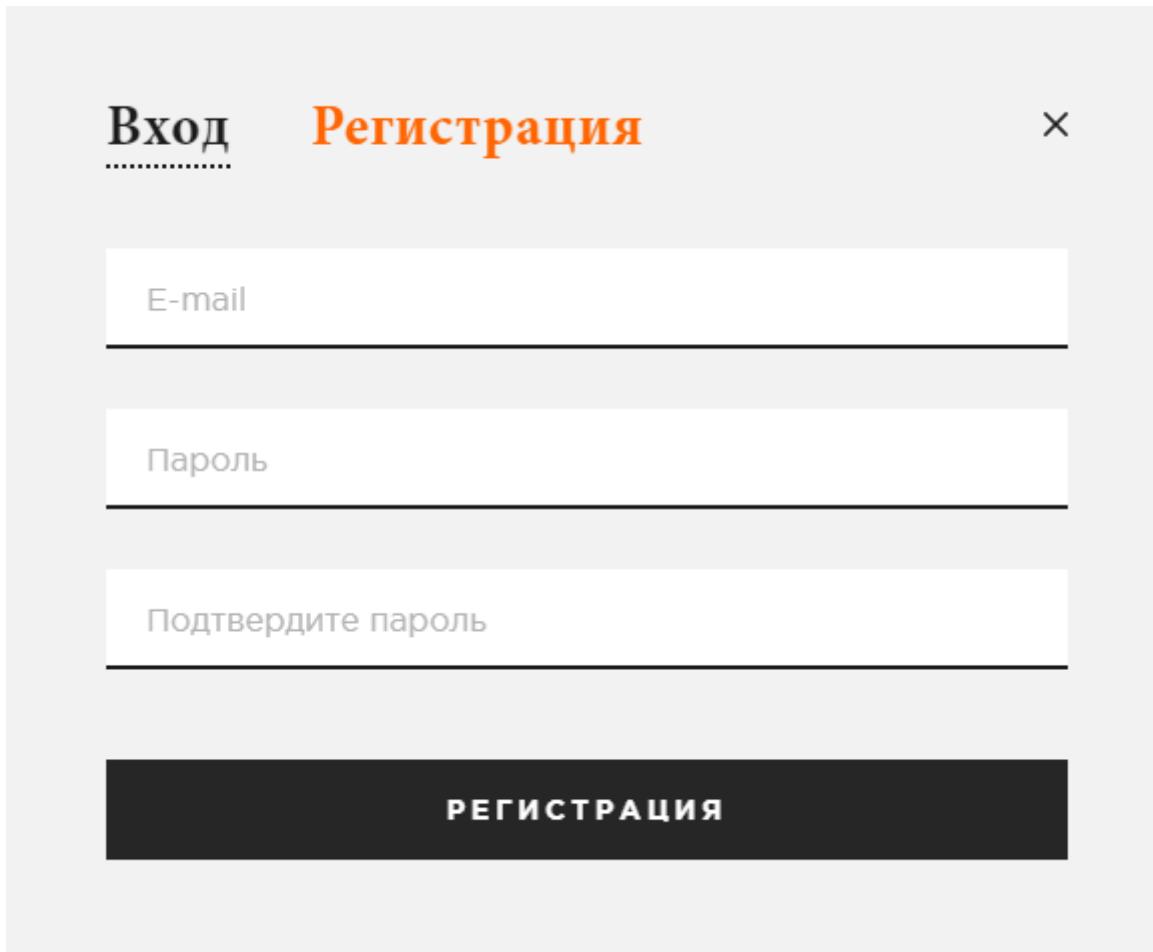


Рисунок 6.2 – Форма регистрации пользователя в системе

В процессе развертывания системы создается учетная запись администратора, для того чтобы иметь возможность сконфигурировать систему с ее использованием. Параметры учетной записи администратора могут быть сконфигурированы в файле конфигурации приложения.

Для регистрации в системе, необходимо ввести адрес электронной почты, пароль и подтверждение пароля. По нажатию на кнопку «Регистрация» будет осуществлено создание новой учетной записи. В случае успешного завершения процесса будет осуществлен автоматический вход в систему, а пароль будет подвержен шифрованию. После регистрации данные для входа будут присланы пользователю на введенный адрес электронной почты. Для изменения данных пользователю необходимо войти в личный кабинет, нажав на соответствующую кнопку, находящуюся в шапке интерфейса и изменить доступные для изменения данные.

6.2.3 Управление программным средством

При успешном прохождении авторизации администратора в системе будет отображена главная страница приложения. На данной странице расположены ссылки на основные части приложения, такие как пользователи, каталог, категории, размеры, цвета, составы сырья продукции. Страница администратора представлена на рисунке 6.3.

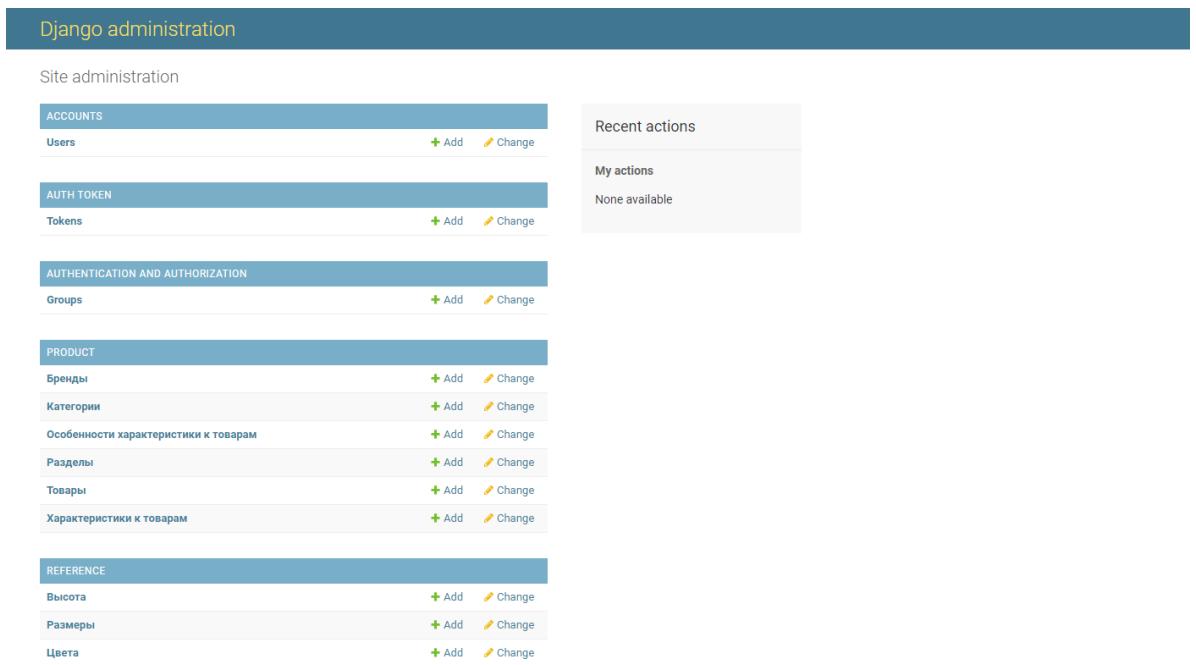


Рисунок 6.3 – Главная страница администратора

6.2.4 Управление пользователями

Нажатие на кнопку «Users» приводит к переходу на страницу со списковой формой пользователей. Данная форма содержит перечисление, всех пользователей, зарегистрированных в системе. Для каждого из существующих пользователей отображаются его адрес электронной почты, фамилия и имя, а также тип учетной записи.

Администратор может осуществить наделение любого их этик пользователей правами администратора, а также блокировать пользователей, установив «Active» в неактивное положение. Наделение пользователя определенными функциями производится путем выбора этих функций, разделенных на следующие категории: accounts, admin, auth, authtoken, contenttypes, product, reference, session. Доступ к каждой категории позволяет пользователю изменять данные, размещенные в этой категории, войдя на форму администратора. На этой же форме возможно и осуществление обратного текущему действия. Пароль пользователя не доступен администратору для изменения, это подтверждается тем, что пароль хранится в зашифрованном состоянии. Внешний вид формы доступен на рисунке 6.4.

Form fields include:

- Last login: Date: Today, Time: Now
- User permissions: accounts | user | Can add user, accounts | user | Can change user, accounts | user | Can delete user, accounts | user | Can log entry, admin | log entry | Can add log entry, admin | log entry | Can change log entry, admin | log entry | Can delete log entry, auth | token | Can refresh token
- First name, Last name
- Staff status, Active (checked)
- Date joined: 2021-01-12, Time: 16:42:27
- Email address

Рисунок 6.4 – Форма изменения данных пользователей в системе.

6.2.5 Просмотр каталога

При успешном прохождении авторизации пользователем в системе ему будет отображена главная страница. На данной странице расположены основные части, такие как фильтры, расположенные слева от основного блока с товарами, виды сортировки, отображенными над товарами, категориями каталога, находящиеся сверху страницы и другие. Пример главной страницы представлен на рисунке 6.3.

Рисунок 6.5 – Просмотр каталога на главной странице

6.2.6 Применение фильтров для просмотра каталогом

Для обеспечения удобства использования программного средства были реализованы возможности сортировки, фильтрации и поиска товаров. Для использования фильтров необходимо выбрать торговую маркой, размер, рост либо цену и дождаться обновления ассортимента товаров.

Сортировка реализована по трем параметрам, это: наименование товара, номер модели и цена. Выбор определенного вида сортировки влечет изменение отображение товаров в порядке «от меньшего к большему».

Также существует два вида отображения блока с товарами – сокращенный и подробный вид. Сокращенный вариант отображения товаров изображен на рисунке 6.6, а подробный вид – на рисунке 6.7.

Рисунок 6.6 – Просмотр каталога с использованием фильтров в сокращенном виде

Рисунок 6.7 – Просмотр каталога с в подробном виде

6.2.7 Поиск товаров

Еще одним фактором, удовлетворяющим требованиям к разрабатывающему программному средству является поиск товаров. Для использования поиска, пользователю необходимо нажать на кнопку со значком «лупа», находящуюся в верхнем правом углу страницы и задать для запроса либо наименование товара, либо же номер модели. Поиск срабатывает при указании двух и более символов. Результат поиска товаров отображен на рисунке 6.8.

The screenshot shows a search results page for the term 'Майка'. The top navigation bar includes links for 'КАТАЛОГ', 'КЛИЕНТАМ', 'КОНТАКТЫ', 'ФРАНЧАЙЗИНГ', and 'ГДЕ КУПИТЬ'. The search bar contains the text 'Майка'. Below the search bar, there are four rows of product cards. Each card displays a small image of a person wearing the item, the product name in Russian, and the model number. The products listed are:

Комплект муж.(майка+трусы)	Пижама жен.(майка+брюки пиж.укор.)	Пижама жен.(майка+брюки пиж.укор.)	Комплект для мал.(майка+трусы)
P204092	P204126	P203381	P202708
Комплект для мал.(майка+трусы)	Комплект для мал.(майка+трусы)	Комплект для мал.(майка+трусы)	Комплект для мал.(майка+трусы)
P202730	P202731	P202732	P202733
Комплект для дев.(майка+трусы)	Комплект для дев.(майка+трусы)	Комплект для дев.(майка+трусы)	Комплект для дев.(майка+трусы)
P202750	P202751	P202752	P202805
Комплект для дев.(майка+трусы)	Комплект для дев.(майка+трусы)	Майка жён.	Комплект для дев.(майка+трусы)
P202806	P202484	P159212	P200636
Комплект для дев.(майка+трусы)	Комплект для дев.(майка+трусы)	Комплект для дев.(майка+трусы)	Комплект для мал.(майка+трусы)
P200637	P200638	P201471	P201472

Рисунок 6.8 – Поиск товара

6.2.8 Просмотр детализации товара

Просмотр всех параметров товара является необходимым для каждого интернет-магазина. Степень описания всевозможных атрибутов товара зависит от проработки интернет-магазина, но обязательными для всех в данной предметной области являются следующие: модель, определяющая тип товара, артикул, являющийся уникальным идентификатором модели, набор допустимых цветовых вариаций модели, таблица размеров-ростов, задающая размеры товара, состав сырья, из которого была пошита данная модель, отношение к определенной торговой марке и отпускную цену. Ниже данного блока отображена секция, позволяющая пользователю выбрать для покупки модель в определенной конфигурации. Для выбора доступна модель в доступных цветовых вариациях, для каждой из которых определена таблица с размерами и ростами. Для выбора достаточно увеличить количество единиц изделий в таблице, после чего будет подсчитана сумма товаров, отображенная внизу данного блока. Текущая страница представлена на рисунках 6.9 и 6.10.

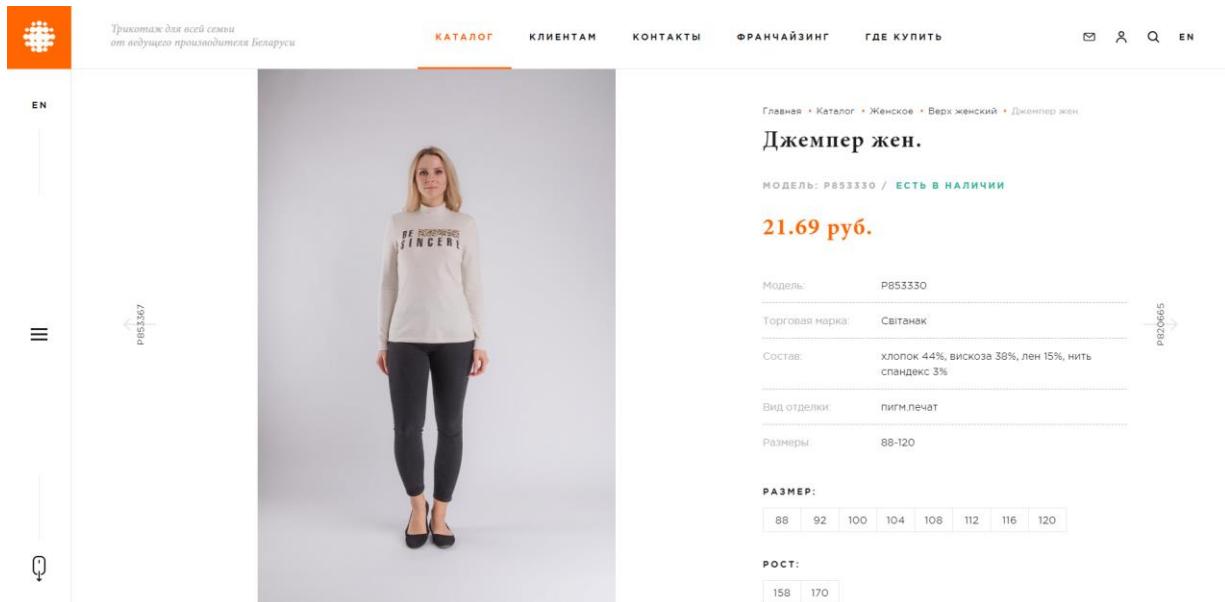


Рисунок 6.9 – Просмотр карточки товара

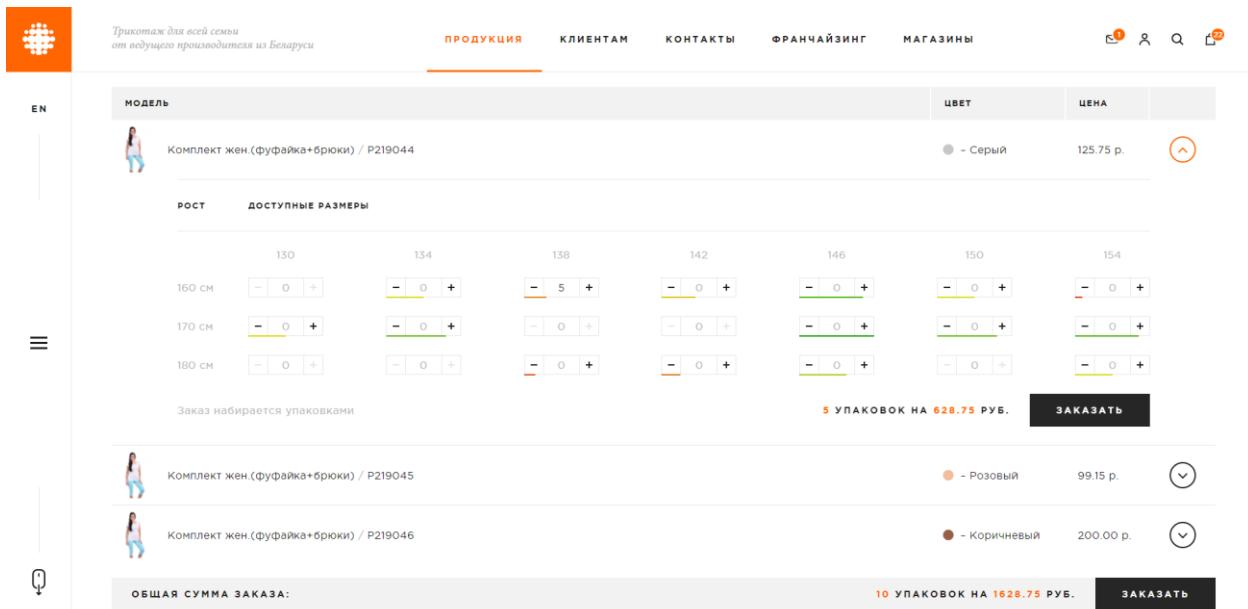


Рисунок 6.10 – Просмотр допустимых комбинаций размеров и ростов

6.2.9 Просмотр детализации товара

Наиболее важной функциональной возможностью, отличающей интернет-магазин от интернет-витрин, является возможность формировать виртуальную корзину. Пример добавления товаров в корзину изображен на рисунке 6.11. Для совершения данного действия пользователю необходимо определить конкретные параметры товара, удовлетворяющие его потребностям и нажать кнопку «Добавить в корзину», находясь на странице детализации товара. После успешного добавления товаров в корзину, счетчик, находящийся в правом верхнем углу, обновится, а при нажатии на него отобразится блок предпросмотра корзины, отображающий список добавленных в корзину товаров.

The screenshot shows a product catalog page for 'Belye Yasel'shoe'. At the top, there are navigation links: 'ПРОДУКЦИЯ', 'КЛИЕНТАМ', 'КОНТАКТЫ', 'ФРАНЧАЙЗИНГ', 'МАГАЗИНЫ', and a search bar. On the left, there are filters for 'НАЗВАНИЕ ИЗДЕЛИЯ', 'НАЛИЧИЕ', and 'ЦЕНА'. The main content area displays a grid of products under categories: 'БЕЛЬЕ ЖЕНСКОЕ ВЕРХ ЖЕНСКИЙ', 'БЕЛЬЕ МУЖСКОЕ ВЕРХ МУЖСКОЙ', and 'БЕЛЬЕ ДЕТСКОЕ ВЕРХ ДЕТСКИЙ'. Below this, there are sections for 'СЕЗОН:' and 'СОРТИРОВКА ПО: BYN'. A promotional banner for 'GEGO' features three models in pajamas. To the right, a list of items is shown with columns for 'ИМЯННОВАНИЕ', 'КОЛ-ВО', 'ЦЕНА', and 'Действия'. The total price at the bottom is '1 250 РУБ.' with a button 'ПЕРЕЙТИ В КОРЗИНУ'.

Рисунок 6.11 – Просмотр корзины

6.2.9 Оформление заказа

Следующий шаг, ведущий пользователя к приобретению товаров это оформление заказа. Для совершения данного действия необходимо после предпросмотра товаров в корзине нажать кнопку «Перейти в корзину». После загрузки страницы пользователю отображена детализация товаров, выбранных на предыдущих этапах, это: наименование, модель, размер, рост, количество товаров, цвет, стоимость одного товара, итоговая стоимость всех товаров в корзине. Содержание данной страницы изображено на рисунке 6.12.

The screenshot shows the 'Корзина' (Cart) page. At the top, there are navigation links: 'ПРОДУКЦИЯ', 'КЛИЕНТАМ', 'КОНТАКТЫ', 'ФРАНЧАЙЗИНГ', 'МАГАЗИНЫ', and a search bar. The main content area shows a table of items with columns: 'НАИМЕНОВАНИЕ', 'МОДЕЛЬ', 'РАЗМЕР', 'РОСТ', 'КОЛ-ВО УП.', 'В УП.', 'ВСЕГО', 'ЦЕНА', 'СТОИМОСТЬ', and a delete icon. The table includes four rows of data. Below the table, a note states: 'Комплект жен. (Фуфайка+брюки)-прошел областной этап республиканского конкурса профессионального мастерства среди швей:'. At the bottom right, there is a note: 'Данный товар удалён из каталога'.

Рисунок 6.12 – Страница оформления заказа

6.2.10 Подтверждение заказа

Завершающим этапом заказа товаров является непосредственное оформление заказа. Для совершения данного действия пользователю необходимо указать личные данные для того, чтобы в последующем менеджер компании смог связаться с покупателем. В случае, если пользователь при регистрации указал все необходимые для оформления заказа данные, повторное указание не требуется, данные будут загружены автоматически. Также пользователю предоставляется возможность указать комментарий к заказу. Пример приведен на рисунке 6.13.

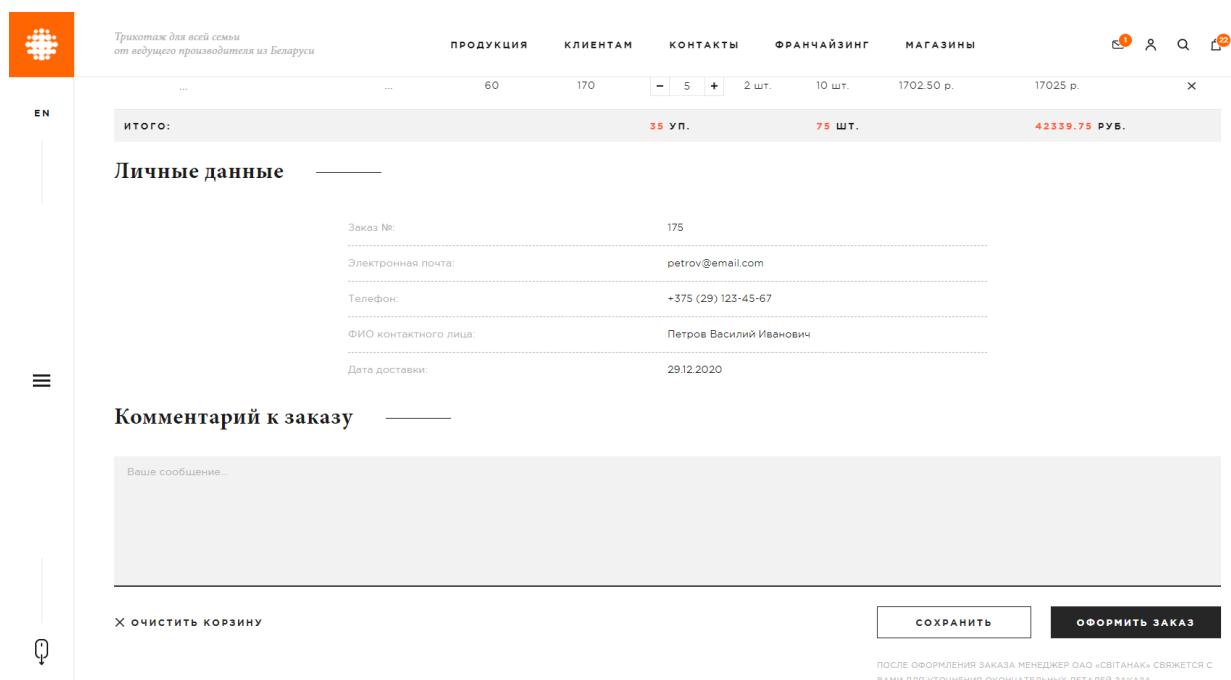


Рисунок 6.13 – Подтверждение заказа

При успешном выполнении процедуры оформления заказа пользователю будет отображено уведомление с информацией об этом. Информация в поле «Дата поставки» обновится после обработки данного заказа менеджером по продажам, который будет сопровождать данный заказ до момента получения его покупателем. Детализация заказа будет отправлена пользователю посредством электронной почты на адрес, указанный пользователем при регистрации.

Таким образом, при разработке интернет-магазина были учтены все нюансы разработки. Приложение интуитивно понятно и удобно в использовании, и не требует дополнительных навыков для его использования.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ИНТЕРНЕТ-МАГАЗИНА ДЛЯ ОАО «СВІТАНАК»

7.1 Описание функций, назначения и потенциальных пользователей программного продукта

Разрабатываемое в дипломном проекте программное средство предоставляет пользователю возможность проведения покупок в режиме онлайн. Также данной программное средство предназначено упростить процессы заказа продукции, которые на текущий момент зачастую выполняются вручную, тем самым освобождая часть рабочего времени работников магазинов и менеджеров по продажам. Пользователем программного средства может выступать любой человек, который имеет персональный компьютер либо мобильное устройство с доступом к сети Интернет.

Данное программное средство разрабатывается собственными силами предприятия для собственных нужд с целью повышения эффективности продаж и снижения временных затрат сотрудников на обработку заказов.

Программное средство будет являться дополнительной площадкой для ведения коммерческой деятельности организации ОАО «Світанак».

7.2 Расчет затрат на разработку программного средства

Задачей данного раздела дипломного проекта является подтверждение актуальности и экономической целесообразности разработки программного средства. Раздел включает в себя следующие пункты:

- затраты на основную заработную плату разработчиков;
- затраты на дополнительную заработную плату разработчиков;
- отчисления на социальные нужды;
- прочие затраты (амortизация оборудования, расходы на электроэнергию, командировочные расходы, накладные расходы и т.п.)

7.2.1 Расчет затрат на основную заработную плату разработчиков

Затраты на основную заработную плату определяются составом команды, которая занимается разработкой программного средства, месячным окладом специалистов и трудоемкостью процесса разработки и рассчитываются по формуле:

$$Z_0 = \sum_{i=1}^n Z_{ci} \cdot t_i, \quad (7.1)$$

где n – количество исполнителей, занятых разработкой конкретного ПО;
 Z_{ci} – часовая заработка i -го исполнителя (руб.);
 t_i – трудоемкость работ, выполняемых i -м исполнителем (ч).

Для разработки данного программного продукта была выбрана команда разработчиков в составе бизнес-аналитика, занимающегося анализом потребностей клиента с их последующим улучшением, написание требований для ПС и их спецификаций, системного архитектора, разрабатывающего и подготавливающего документацию, описание сущностей, взаимосвязей и процессов предметной области, тестирующего, выявляющего ошибки в работе программного средства, дизайнера, создающего интерфейс ПС, а также двух инженеров-программистов, один из которых является ведущим. Причиной этого является то, что конечный продукт должен состоять из двух частей: клиентской части, непосредственно с которой будут работать пользователи, а также серверной части, на которой осуществляется долговременное хранение результатов работы с программным средством и обработка информации. Весьма целесообразной является параллельная разработка обеих частей программного средства, которой можно добиться, поручив клиентскую часть инженеру-программисту, а вторую часть отдать на выполнение ведущему инженеру-программисту, являющемуся специалисту в области разработки серверных решений. Такое разделение позволит закончить проект вовремя с учетом рисков, связанных с разработкой, и выполнить его качественнее благодаря специализации разработчиков.

Расчетная норма рабочего времени принята равной 168 часам. Данные по заработной плате команды разработчиков предоставлены ОАО «Світанак» на 13 ноября 2020 года. Расчет затрат на основную заработную плату осуществлен в форме таблицы 7.1.

Таблица 7.1 – Расчет затрат на основную заработную плату команды разработчиков.

№ п/п	Участник команды	Месячная заработная плата, руб	Часовая заработная плата, руб	Трудоемкость работ, ч	Зарплата по тарифу, руб
1	2	3	4	5	6
1	Бизнес-аналитик	1 890	11,25	16	180
2	Системный архитектор	2 459,52	14,64	24	351,36
3	Ведущий инженер-программист	2 926,56	17,42	56	975,52
4	Инженер-программист	2 640,96	15,72	120	1 886,4
5	Тестировщик	1 990,8	11,85	16	189,6
6	Дизайнер	1 668,24	9,93	24	238,32
Премия (50%)					1 910,6
Итого затраты за основную зарплату разработчиков					5 731,8

7.2.2 Расчет затрат на дополнительную заработную плату
Дополнительная заработка исполнителей проекта., определяется по формуле:

$$Z_d = \frac{Z_o \cdot H_d}{100}, \quad (7.2)$$

где Z_o – затраты на основную заработную плату, (руб.);
 H_d – норматив дополнительной заработной платы (15%).

Таким образом, затраты на дополнительную заработную плату составят:

$$Z_d = \frac{5\,731,8 * 15}{100} = 859,77 \text{ (руб)}$$

7.2.3 Расчет отчислений на социальные нужды

Отчисления в фонд социальной защиты населения и на обязательное страхование определяются в соответствии с действующими законодательными актами по формуле:

$$Z_{cz} = \frac{(Z_o + Z_d) \cdot H_{coz}}{100}, \quad (7.3)$$

где Z_o – затраты на основную заработную плату, (руб.);
 Z_d – затраты на дополнительную заработную плату, (руб.)
 H_{coz} – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34,6%).

Таким образом, отчисления в фонд социальной защиты населения и на обязательное страхование составят:

$$Z_{cz} = \frac{(5\,731,8 + 859,77) \cdot 34,6}{100} = 2\,280,68 \text{ (руб)}$$

7.2.4 Расчет прочих затрат

Прочие затраты включают затраты, связанные с разработкой конкретного программного обеспечения напрямую, а также связанные с функционированием организации-разработчика в целом. Расчет прочих затрат выполняется в процентах от затрат на основную заработную плату команды разработчиков с учетом премии по формуле:

$$Z_{pz} = \frac{Z_o \times H_{pz}}{100}, \quad (7.4)$$

где Z_o – затраты на основную заработную плату, (руб.);

$H_{пз}$ – норматив прочих затрат (125%).

Таким образом:

$$Z_{пз} = \frac{5\ 731,8 * 125}{100} = 6\ 878,16 \text{ (руб)}$$

Полная сумма затрат на разработку программного обеспечения находится путем суммирования всех рассчитанных статей затрат. Расчет приведен в таблице 7.2.

Таблица 7.2 – Затраты на разработку программного обеспечения.

Статья затрат	Сумма, руб
1	2
Основная заработка команда разработчиков	5 731,8
Дополнительная заработка команда разработчиков	859,77
Отчисления на социальные нужды	2 280,68
Прочие затраты	6 878,16
Общая сумма затрат на разработку	15 750,41

Рассчитанное значение полной себестоимости, которое составило 15750,41 руб., будет использоваться в дальнейшем для определения экономического эффекта, а также уровня рентабельности разработки и внедрения программного средства.

7.3 Оценка экономического эффекта от использования ПС

Разрабатываемое программное средство предоставляет экономический эффект. На консультацию покупателя время у маркетолога примерно составляет 10 минут, что в процентном соотношении от часа примерно равно 15%. Среднее время, затрачиваемое на консультацию клиента и дистанционный подбор товара с использованием программного средства, сократится примерно на 30% для одного заказа.

В двух отделах розничной торговли в сумме работает 12 менеджеров по продажам, для которых экономия выразится в освобождении времени на консультации по телефону и предоставление подробной информации о товарах посредством электронной почты, что позволит увеличить количество обрабатываемых заказов.

Учитывая, что использование программного средства позволит сэкономить приблизительно 10 минут для каждого сотрудника, то в течение года эта величина составит:

$$(0,15 \cdot 21 \cdot 12) \cdot (8 \cdot 12) = 3\ 628,8 \text{ (ч)}$$

где 0,15 – процент сэкономленного за час времени;

8 – продолжительность рабочего дня в часах;
 21 – среднее количество рабочих дней в месяце;
 12 – количество месяцев в году;
 12 – количество менеджеров по продажам.

Часовая заработная плата менеджера по продажам, по данным предприятия, составляет 10,2 руб.

На основе сэкономленного времени вычислим экономию затрат:

$$\mathcal{E}_3 = 10,2 \cdot 3628,8 = 37\,013,76(\text{руб})$$

Рассчитаем из полученных данных отчисления в ФСЗН и получим следующее:

$$Z_{cz} = \frac{(37\,013,76 + 5\,552,06) \cdot 34,6}{100} = 14\,727,78 \text{ (руб)},$$

а экономия затрат после вычета отчислений составит:

$$\mathcal{E}_3 = 37\,013,76 - 14\,727,78 = 22\,285,98 \text{ (руб)}$$

Годовая экономия текущих затрат составит 22 285,98 рублей.

Экономический эффект рассчитывается по следующей формуле:

$$\Delta\Pi_q = (\mathcal{E}_3 - \Delta Z_{tek}) \cdot (1 - H_p), \quad (7.5)$$

где \mathcal{E}_3 - экономия текущих затрат, полученная в случае применения программного средства, руб;

ΔZ_{tek} – прирост текущих затрат, связанных с использованием программного средства, руб;

H_p – ставка налога на прибыль в соответствии с действующим законодательством, %.

Таким образом, экономический эффект составит:

$$\Delta\Pi_q = (22\,285,98 - 15\,750,41) \cdot (1 - 0,18) = 5\,359,16 \text{ (руб)}$$

Уровень рентабельности рассчитывается по формуле 7.6:

$$y_p = \frac{\Pi(\Pi_q)}{Z_p} \cdot 100\%, \quad (7.6)$$

где Π – прибыль, получаемая от реализации данного ПС (руб.);
 Z_p – общая сумма затрат на разработку ПС (руб.).

Рассчитаем показатель рентабельности:

$$y_p = \frac{5\ 359,16}{15\ 750,41} \cdot 100\% = 34\%$$

По данным затрат на разработку 15 750,41 руб., экономия затрат составит 5 359,16 руб. При этом уровень рентабельности разработки и внедрения программного средства составляет 34%.

ЗАКЛЮЧЕНИЕ

В ходе дипломного проектирования проведен анализ существующих программных решений, изучены наиболее известные решения в сфере Интернет-торговли. В процессе работы были исследованы и выделены основные современные тенденции в области электронной коммерции. Для каждого из упомянутых в отчете программных средств выявлены достоинства и недостатки использования, которые учитывались при разработке спецификации требований к программному средству.

Также в ходе моделирования предметной области была разработана функциональная и информационная модели программного обеспечения. На основе разработанных моделей сформулирована функциональная спецификация к программному обеспечению.

Исходя из полученных на этапе моделирования данных, была спроектирована архитектура программного решения, которая смогла бы позволить точно реализовать требования, указанные в функциональной спецификации. Также на данном этапе были определены архитектурные принципы, в соответствие с которыми должна производиться разработка программного решения.

На основании информационной модели была спроектирована модель базы данных. Данная модель призвана покрыть все необходимые аспекты, связанные с хранением сущностей, используемых внутри системы.

Большую часть времени дипломного проектирования заняло написание клиентской части приложения.

Уделено внимание технико-экономическому обоснованию. Разработано руководство пользователя с подробным описанием использования программного продукта.

В рамках дипломного проекта были углублены знания по разработке мобильных приложений, использованию сервисов по аналитике сторонних приложений, существующих платформ, а также по проектированию пользовательского интерфейса.

Таким образом, задачи, поставленные в рамках индивидуального задания, были выполнены. Знания и опыт полученные в процессе прохождения дипломного проектирования будут полезны при дальнейшей работе по специальности.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- [1] Антиплагиат – Система для онлайн проверки текста на заимствование. [Электронный ресурс]. – Режим доступа: <https://users.antiplagiat.ru/report/byLink/short/1?v=1&userId=8654497&validationHash=88325D4B1B14D6BA884852D0BBCDE5AF103D7C65&c=0> Дата доступа: 06.01.2021.
- [2] ГОСТ 34.003-90. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения. // Информационная технология. Комплекс стандартов и руководящих документов на автоматизированные системы. – М.: Изд-во стандартов, 1991.
- [3] Appendix D. PostgreSQL 9 Documentation. [Электронный ресурс]. – 2009 – Режим доступа: <https://www.postgresql.org/docs/> Дата доступа: 20.11.2020
- [4] Реляционные базы данных обречены? [Электронный ресурс]. – 2016 – Режим доступа: habrahabr.ru/post/103021/ Дата доступа: 25.11.2020
- [5] Django FAQ about MVC in Django [Электронный ресурс]. – 2020 – Режим доступа: <https://docs.djangoproject.com/en/dev/faq/general/#django-appears-to-be-a-mvc/> Дата доступа: 06.12.2020
- [6] Хассан, Г. UML-проектирование систем в приложениях / Г. Хассан – М.: ДМК Пресс, 2011 – 704с.
- [7] Дипломные проекты (работы) общие требования СТП 01–2010 [Электронный ресурс]: стандарт предприятия / БГУИР – Электронные данные. – Режим доступа: СТП П2010 бгуир.pdf
- [8] Технико-экономическое обоснование дипломных проектов: Методическое пособие для студентов БГУИР. В 4-ч. Ч. 4: Проекты программного обеспечения / В.А. Палицын. – Минск: БГУИР, 2006 г. – 76 с.
- [9] Statista – Business Data Platform [Электронный ресурс] 2017 – Режим доступа: <https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/> Дата доступа: 23.12.2020
- [10] Миковски, М. Разработка одностраничных веб-приложений / Майкл Миковски, Джош Пауэлл – ДМК Пресс, 2014. – 512 с.
- [11] Nuxt.js: Universal Vue.js Apps [Электронный ресурс] – 2019 – Режим доступа: <https://www.sitepoint.com/nuxt-js-vue-js/> Дата доступа: 13.12.2020
- [12] Veore.by – Интернет магазин одежды и обуви в Беларуси - Veore Clothing [Электронный ресурс] – 2020 – Режим доступа <https://veore.by/> Дата доступа: 08.11.2020
- [13] ConteShop.by – Интернет-магазин одежды белорусских производителей – [Электронный ресурс] – 2020 – Режим доступа: <https://conteshop.by/> Дата доступа: 08.11.2020
- [14] Kupalinka.com – ОАО "Купалинка" – [Электронный ресурс] – 2020 – Режим доступа: <http://www.kupalinka.com/> Дата доступа: 08.11.2020
- [15] Майерс, Г. Искусство тестирования программ, 3-е издание / Гленфорд Майерс, Том Баджетт, Кори Сандлер. – М.: «Диалектика», 2012. – 272 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Текст программного средства

Серверная часть программного средства:

```
import os
import sys

def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'svitanak.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()

from django.core.exceptions import MultipleObjectsReturned
from django.contrib.auth.backends import ModelBackend
from django.contrib.auth import get_user_model
from django.db.models import Q

UserModel = get_user_model()

class EmailBackend(ModelBackend):
    def authenticate(self, request, username=None, password=None, **kwargs):
        try:
            user = UserModel.objects.get(email=username)
        except UserModel.DoesNotExist:
            UserModel().set_password(password)
        except MultipleObjectsReturned:
            return UserModel.objects.filter(email=username).order_by('id').first()
        else:
            if user.check_password(password) and self.user_can_authenticate(user):
                return user

    def get_user(self, user_id):
        try:
```

```

        user = UserModel.objects.get(pk=user_id)
    except UserModel.DoesNotExist:
        return None

    return user if self.user_can_authenticate(user) else None

from django.db import models
from django.contrib.auth.models import AbstractUser
from django.utils.translation import gettext_lazy as _
from django.contrib.auth.base_user import BaseUserManager

class UserManager(BaseUserManager):

    def create_user(self, email, password=None):

        if email is None:
            raise TypeError('Users must have an email address.')

        user = self.model(email=self.normalize_email(email))
        user.set_password(password)
        user.save()

        return user

    def create_superuser(self, email, password):

        if password is None:
            raise TypeError('Superusers must have a password.')

        user = self.create_user(email, password)
        user.is_superuser = True
        user.is_staff = True
        user.save()

        return user

class CustomUser(AbstractUser):
    username = None

    email = models.EmailField(_('email address'), unique=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    objects = UserManager()

from rest_framework import serializers

from django.contrib.auth import get_user_model

```

```

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = get_user_model()
        fields = ('email', 'password')
        extra_kwargs = {
            'password': {'write_only': True},
        }

import re

from django.core.mail import send_mail
from django.conf import settings

from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.status import HTTP_400_BAD_REQUEST

class EmailMessageView(APIView):

    def post(self, request):
        data = request.data
        print(data)
        first_name = data.get('first_name')
        email = data.get('email')
        phone = data.get('phone')
        message = data.get('message')

        if not first_name:
            return Response("Имя не было заполнено.", status=HTTP_400_BAD_REQUEST)

        if not isinstance(email, str):
            return Response("Email не был указан.", status=HTTP_400_BAD_REQUEST)
        elif isinstance(email, str) and not re.match(r"^\+@(\[?\) [a-zA-Z0-9-\.]+\.\([a-zA-Z]\{2,3\}\| [0-9]\{1,3\}\) (\[?\)$", email):
            return Response("Email был указан неверно.", status=HTTP_400_BAD_REQUEST)

        if not isinstance(phone, str):
            return Response("Телефон не был указан.", status=HTTP_400_BAD_REQUEST)
        elif isinstance(phone, str) and not re.match(r"(\+375)\?(\s*)\?(\d{9})", phone):
            return Response("Телефон был указан неверно.", status=HTTP_400_BAD_REQUEST)
        if not message:
            return Response("Сообщение не было заполнено.", status=HTTP_400_BAD_REQUEST)

        data = (
            "Поступила новая заявка\n"

```

```

        f"Имя: {first_name}\n"
        f"Email: {email}\n"
        f"Телефон: {phone}\n"
        f"Текст сообщения: {message}\n"
    )

    send_mail(
        'Новая заявка',
        data,
        settings.EMAIL_HOST_USER,
        settings.EMAILS_FOR_MESSAGES,
        fail_silently=False
    )

    return Response(status=200)

class PaginationMixin(object):

    @property
    def paginator(self):
        """
        The paginator instance associated with the view, or None.
        """
        if not hasattr(self, '_paginator'):
            if self.pagination_class is None:
                self._paginator = None
            else:
                self._paginator = self.pagination_class()
        return self._paginator

    def paginate_queryset(self, queryset):
        """
        Return a single page of results, or None if pagination is disabled.
        """
        if self.paginator is None:
            return None
        return self.paginator.paginate_queryset(queryset, self.request,
                                                view=self)

    def get_paginated_response(self, data):
        """
        Return a paginated style 'Response' object for the given output data.
        """
        assert self.paginator is not None
        return self.paginator.get_paginated_response(data)

from rest_framework.pagination import PageNumberPagination
from rest_framework.response import Response

from django.conf import settings

```

```

class StandardResultsPagination(PageNumberPagination):
    page_size = settings.PAGE_SIZE
    page_size_query_param = 'limit'
    max_page_size = settings.PAGE_SIZE

    def get_paginated_response(self, data):
        return Response({
            'links': {
                'has_next': bool(self.get_next_link()),
                'has_prev': bool(self.get_previous_link())
            },
            'count': self.page.paginator.count,
            'results': data,
            'total_pages': self.page.paginator.num_pages,
            'current_page': self.page.number,
        })

from django.db.models import Q

from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.status import HTTP_404_NOT_FOUND

from api.pagination import StandardResultsPagination
from api.mixins import PaginationMixin

from product.models import (
    Category,
    Product,
    ProductFeatureItem,
    Brand,
    ProductFeature,
)
from reference.models import (
    ProductSize,
    ProductHeight,
    ProductColor,
)
from product.serializers import (
    CategorySerializer,
    ProductShortSerializer,
    BrandSerializer,
    ProductSerializer,
)
from reference.serializers import (
    ProductSizeSerializer,
    ProductHeightSerializer,
    ProductColorSerializer,
)

```

```

NAME_COLOR = 'colors'
NAME_BRAND = 'brands'
NAME_SIZE = 'sizes'
NAME_HEIGHT = 'height'
NAME_CATEGORY = 'category'

PRODUCTS_ORDER_BY = {
    '1': {'value': 'name', 'title': "по названию"},
    '2': {'value': 'model', 'title': "по модели"},
    '3': {'value': 'price', 'title': "по цене"},
}

class InitializeDataView(APIView):

    def get(self, request, format=None):
        query_params = self.request.query_params
        only_models = query_params.get('only', str()).split(',')

        categories = Category.objects.all()

        orders = {key: PRODUCTS_ORDER_BY.get(key).get('title') for key in
PRODUCTS_ORDER_BY.keys()}

        result = {
            'categories': CategorySerializer(categories, many=True).data,
            'orders': orders
        }

        if NAME_BRAND in only_models:
            brands = Brand.objects.all()
            result.update({
                'brands': BrandSerializer(brands, many=True).data
            })

        if NAME_COLOR in only_models:
            colors = ProductColor.objects.all()
            result.update({
                'colors': ProductColorSerializer(colors, many=True).data
            })

        if NAME_HEIGHT in only_models:
            height = ProductHeight.objects.all()
            result.update({
                'height': ProductHeightSerializer(height, many=True).data
            })

        if NAME_SIZE in only_models:
            sizes = ProductSize.objects.all()
            result.update({
                'sizes': ProductSizeSerializer(sizes, many=True).data
            })

```

```

    return Response(result)

class ProductsView(APIView, PaginationMixin):
    pagination_class = StandardResultsPagination
    serializer_class = ProductShortSerializer

    def get(self, request, format=None):
        filters, product_features_items, product_features = self._generate_filter(request.query_params)
        queryset = Product.objects.all().filter(**filters)
        if product_features_items:
            queryset = queryset.filter(
                id__in=product_features_items.values_list(
                    'product_feature_product_id', flat=True
                ),
            )

        if product_features:
            queryset = queryset.filter(
                id__in=product_features.values_list(
                    'product_id', flat=True
                ),
            )

        search_value = request.query_params.get('search')
        if search_value:
            queryset = queryset.filter(
                Q(name__icontains=search_value) | Q(model__icon__contains=search_value)
            )

        order_by = self._generate_order_by(request.query_params)
        queryset = queryset.order_by(order_by)

        paginated = dict()
        page = self.paginate_queryset(queryset)

        if page is not None:
            serializer_products = ProductShortSerializer(page, many=True)
            paginated = self.get_paginated_response(serializer_products.data)
            return paginated
        return Response(paginated)

    @staticmethod
    def _generate_filter(params):
        filters = {}

        is_new = params.get('only_new', None)
        if is_new is not None:
            filters.update({
                'is_new': bool(params.get('only_new', False)),
            })

```

```

brands = params.get(NAME_BRAND)
if brands:
    brands = [brand for brand in brands.split(',') if
brand.isdigit()]
    filters.update({'brand_id_in': brands})

products_feature_items = {}
product_feature = {}

colors = params.get(NAME_COLOR)
if colors:
    colors = [color for color in colors.split(',') if
color.isdigit()]
    product_feature.update({
        'color_id_in': colors,
    })

heights = params.get(NAME_HEIGHT)
if heights:
    heights = [height for height in heights.split(',') if
height.isdigit()]
    products_feature_items.update({
        'height_id_in': heights,
    })

sizes = params.get(NAME_SIZE)
if sizes:
    sizes = [size for size in sizes.split(',') if size.isdigit()]
    products_feature_items.update({
        'size_id_in': sizes,
    })

subcategory = params.get(NAME_CATEGORY)
if subcategory:
    subcategories = [subcategory_ for subcategory_ in subcate-
gory.split(',') if subcategory_.isdigit()]
    filters.update({
        'subcategory_id_in': subcategories
    })

if products_feature_items:
    products_features_items = ProductFeatureItem.objects.fil-
ter(**products_feature_items)
else:
    products_features_items = ProductFeatureItem.objects.none()

if product_feature:
    product_features = ProductFeature.objects.filter(**product_fea-
ture)
else:
    product_features = ProductFeature.objects.none()

```

```

        return filters, products_features_items, product_features

    @staticmethod
    def _generate_order_by(params):
        default_value = '-id'
        return PRODUCTS_ORDER_BY.get(params.get('order_by', None),
        {}).get('value', default_value)

class ProductView(APIView):
    serializer_class = ProductShortSerializer

    def get(self, request, id, format=None, ):
        try:
            product = Product.objects.get(id=id)
        except Product.DoesNotExist:
            return Response({}, status=HTTP_404_NOT_FOUND)
        data = ProductSerializer(product, many=False).data

        try:
            next_product = Product.objects.get(id=id + 1)
        except Product.DoesNotExist:
            next_product = None

        try:
            prev_product = Product.objects.get(id=id - 1)
        except Product.DoesNotExist:
            prev_product = None

        if next_product:
            next_product = {'id': id + 1, 'image': next_product.image.url,
'model': next_product.model}
        if prev_product:
            prev_product = {'id': id - 1, 'image': prev_product.image.url,
'model': prev_product.model}

        list_ = []

        for p in ProductFeature.objects.filter(product=product).distinct('color'):
            items = ProductFeatureItem.objects.filter(product_feature=p)
            table, heights, sizes = get_table_from_features(items, product)
            list_.append({
                'id': p.id,
                'data': table,
                'sizes': sizes,
                'heights': heights,
                'color': ProductColorSerializer(p.color, many=False).data,
            })

        print(list_)

        data.update({

```

```

        'next_product': next_product,
        'prev_product': prev_product,
        'in_stock': ProductFeatureItem.objects.filter(product_feature__product=product, count__gte=0).exists(),
        'feature_items': list_
    })

    return Response(data)

def get_table_from_features(items, product):
    heights = items.distinct('height').values_list('height_value', flat=True)
    sizes = items.distinct('size').values_list('size_value', flat=True)

    data = {index: [] for index, i in enumerate(heights)}

    for index, i in enumerate(heights):
        for s in sizes:
            try:
                item = items.get(height_value=i, size_value=s)
                obj = {
                    'count': item.count,
                    'id': item.id,
                    'value_': 0
                }
            except ProductFeatureItem.DoesNotExist:
                obj = None

            data[index].append(obj)

    return data, heights, sizes

from django.db.models import Q

from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.status import HTTP_404_NOT_FOUND

from api.pagination import StandardResultsPagination
from api.mixins import PaginationMixin

from product.models import (
    Category,
    Product,
    ProductFeatureItem,
    Brand,
    ProductFeature,
)
from reference.models import (
    ProductSize,
    ProductHeight,

```

```

        ProductColor,
    )

from product.serializers import (
    CategorySerializer,
    ProductShortSerializer,
    BrandSerializer,
    ProductSerializer,
)

from reference.serializers import (
    ProductSizeSerializer,
    ProductHeightSerializer,
    ProductColorSerializer,
)

NAME_COLOR = 'colors'
NAME_BRAND = 'brands'
NAME_SIZE = 'sizes'
NAME_HEIGHT = 'height'
NAME_CATEGORY = 'category'

PRODUCTS_ORDER_BY = {
    '1': {'value': 'name', 'title': "по названию"},
    '2': {'value': 'model', 'title': "по модели"},
    '3': {'value': 'price', 'title': "по цене"},
}
}

class InitializeDataView(APIView):

    def get(self, request, format=None):
        query_params = self.request.query_params
        only_models = query_params.get('only', str()).split(',')

        categories = Category.objects.all()

        orders = {key: PRODUCTS_ORDER_BY.get(key).get('title') for key in
                  PRODUCTS_ORDER_BY.keys()}

        result = {
            'categories': CategorySerializer(categories, many=True).data,
            'orders': orders
        }
        if NAME_BRAND in only_models:
            brands = Brand.objects.all()
            result.update({
                'brands': BrandSerializer(brands, many=True).data
            })

        if NAME_COLOR in only_models:
            colors = ProductColor.objects.all()
            result.update({

```

```

        'colors': ProductColorSerializer(colors, many=True).data
    })

    if NAME_HEIGHT in only_models:
        height = ProductHeight.objects.all()
        result.update({
            'height': ProductHeightSerializer(height, many=True).data
        })

    if NAME_SIZE in only_models:
        sizes = ProductSize.objects.all()
        result.update({
            'sizes': ProductSizeSerializer(sizes, many=True).data
        })

    return Response(result)

class ProductsView(APIView, PaginationMixin):
    pagination_class = StandardResultsPagination
    serializer_class = ProductShortSerializer

    def get(self, request, format=None):
        filters, product_features_items, product_features = self._generate_filter(request.query_params)
        queryset = Product.objects.all().filter(**filters)
        if product_features_items:
            queryset = queryset.filter(
                id__in=product_features_items.values_list(
                    'product_feature_product_id', flat=True
                ),
            )
        if product_features:
            queryset = queryset.filter(
                id__in=product_features.values_list(
                    'product_id', flat=True
                ),
            )
        search_value = request.query_params.get('search')
        if search_value:
            queryset = queryset.filter(
                Q(name__icontains=search_value) | Q(model__icontains=search_value)
            )
        order_by = self._generate_order_by(request.query_params)
        queryset = queryset.order_by(order_by)

        paginated = dict()
        page = self.paginate_queryset(queryset)

```

```

if page is not None:
    serializer_products = ProductShortSerializer(page, many=True)
    paginated = self.get_paginated_response(serializer_products.data)
    return paginated
return Response(paginated)

@staticmethod
def _generate_filter(params):
    filters = {}

    is_new = params.get('only_new', None)
    if is_new is not None:
        filters.update({
            'is_new': bool(params.get('only_new', False)),
        })

    brands = params.get(NAME_BRAND)
    if brands:
        brands = [brand for brand in brands.split(',') if
brand.isdigit()]
        filters.update({'brand_id_in': brands})

    products_feature_items = {}
    product_feature = {}

    colors = params.get(NAME_COLOR)
    if colors:
        colors = [color for color in colors.split(',') if
color.isdigit()]
        product_feature.update({
            'color_id_in': colors,
        })

    heights = params.get(NAME_HEIGHT)
    if heights:
        heights = [height for height in heights.split(',') if
height.isdigit()]
        products_feature_items.update({
            'height_id_in': heights,
        })

    sizes = params.get(NAME_SIZE)
    if sizes:
        sizes = [size for size in sizes.split(',') if size.isdigit()]
        products_feature_items.update({
            'size_id_in': sizes,
        })

    subcategory = params.get(NAME_CATEGORY)
    if subcategory:
        subcategories = [subcategory_ for subcategory_ in subcate-
gory.split(',') if subcategory_.isdigit()]
        filters.update({

```

```

        'subcategory__id__in': subcategories
    })

    if products_feature_items:
        products_features_items = ProductFeatureItem.objects.filter(**products_feature_items)
    else:
        products_features_items = ProductFeatureItem.objects.none()

    if product_feature:
        product_features = ProductFeature.objects.filter(**product_feature)
    else:
        product_features = ProductFeature.objects.none()

    return filters, products_features_items, product_features

@staticmethod
def _generate_order_by(params):
    default_value = '-id'
    return PRODUCTS_ORDER_BY.get(params.get('order_by', None),
{}).get('value', default_value)

class ProductView(APIView):
    serializer_class = ProductShortSerializer

    def get(self, request, id, format=None, ):
        try:
            product = Product.objects.get(id=id)
        except Product.DoesNotExist:
            return Response({}, status=HTTP_404_NOT_FOUND)
        data = ProductSerializer(product, many=False).data

        try:
            next_product = Product.objects.get(id=id + 1)
        except Product.DoesNotExist:
            next_product = None

        try:
            prev_product = Product.objects.get(id=id - 1)
        except Product.DoesNotExist:
            prev_product = None

        if next_product:
            next_product = {'id': id + 1, 'image': next_product.image.url,
'model': next_product.model}
            if prev_product:
                prev_product = {'id': id - 1, 'image': prev_product.image.url,
'model': prev_product.model}

        list_ = []

```

```

        for p in ProductFeature.objects.filter(product=product).distinct('color'):
            items = ProductFeatureItem.objects.filter(product_feature=p)
            table, heights, sizes = get_table_from_features(items, product)
            list_.append({
                'id': p.id,
                'data': table,
                'sizes': sizes,
                'heights': heights,
                'color': ProductColorSerializer(p.color, many=False).data,
            })

    print(list_)

    data.update({
        'next_product': next_product,
        'prev_product': prev_product,
        'in_stock': ProductFeatureItem.objects.filter(product_feature__product=product, count__gte=0).exists(),
        'feature_items': list_
    })

    return Response(data)

def get_table_from_features(items, product):
    heights = items.distinct('height').values_list('height__value', flat=True)
    sizes = items.distinct('size').values_list('size__value', flat=True)

    data = {index: [] for index, i in enumerate(heights)}

    for index, i in enumerate(heights):
        for s in sizes:
            try:
                item = items.get(height__value=i, size__value=s)
                obj = {
                    'count': item.count,
                    'id': item.id,
                    'value_': 0
                }
            except ProductFeatureItem.DoesNotExist:
                obj = None

            data[index].append(obj)

    return data, heights, sizes

import uuid

from django.db import models
from django.contrib.auth import get_user_model

```

```

class Card(models.Model):
    session_id = models.UUIDField(default=uuid.uuid4, editable=False,
unique=True)
    user = models.ForeignKey(get_user_model(), null=True)

    products = models.ManyToManyField('products.Product')

class Order(models.Model):
    user = models.ForeignKey(get_user_model())
    products = models.ManyToManyField('products.Product')

from django.db import models

from reference.models import (
    ProductSize,
    ProductHeight,
    ProductColor,
)

class Category(models.Model):
    """
    Модель раздела
    """
    name = models.CharField(
        max_length=64,
        verbose_name="Название раздела",
    )
    image = models.FileField(null=True)

    class Meta:
        verbose_name = "Раздел"
        verbose_name_plural = "Разделы"

    def __str__(self):
        return self.name

class Subcategory(models.Model):
    """
    Модель категории
    """
    category = models.ForeignKey(
        Category,
        on_delete=models.CASCADE,
        verbose_name="Раздел",
    )

    name = models.CharField(
        max_length=64,
        verbose_name="Название категории",
    )

```

```

class Meta:
    verbose_name = "Категория"
    verbose_name_plural = "Категории"

def __str__(self):
    return self.name


class Brand(models.Model):
    """
    Модель бренда
    """

    name = models.CharField(
        max_length=64,
        verbose_name="Наименование бренда"
    )
    image = models.FileField(null=True)

    class Meta:
        verbose_name = "Бренд"
        verbose_name_plural = "Бренды"

    def __str__(self):
        return self.name


class Product(models.Model):
    """
    Модель товара
    """

    subcategory = models.ForeignKey(
        Subcategory,
        on_delete=models.SET_NULL,
        null=True,
        verbose_name="Категория",
    )
    name = models.CharField(
        max_length=268,
        verbose_name="Наименование товара"
    )
    model = models.CharField(
        max_length=16,
        unique=True,
        verbose_name="Модель товара"
    )
    brand = models.ForeignKey(
        Brand,
        on_delete=models.SET_NULL,
        null=True,
        verbose_name="Бренд"
    )

```

```

is_new = models.BooleanField(
    default=False,
    verbose_name="Новинка?"
)

price = models.FloatField(
    verbose_name="Стоимость за шт."
)

image = models.FileField('Изображение', default=None)

class Meta:
    verbose_name = "Товар"
    verbose_name_plural = "Товары"

def __str__(self):
    return f'{self.name} / {self.id}'

class ProductFeature(models.Model):
    """
    Модель характеристик товара
    """
    product = models.ForeignKey(
        Product,
        on_delete=models.CASCADE,
        verbose_name="Товар",
    )
    color = models.ForeignKey(
        ProductColor,
        on_delete=models.PROTECT,
        verbose_name="Цвет",
        null=True,
    )

    class Meta:
        verbose_name = "Характеристика к товару"
        verbose_name_plural = "Характеристики к товарам"

    def __str__(self):
        return f'Характеристика к товару {str(self.product)}'

class ProductFeatureItem(models.Model):
    """
    Модель характеристики товара
    """
    product_feature = models.ForeignKey(
        ProductFeature,
        on_delete=models.CASCADE,
    )
    height = models.ForeignKey(
        ProductHeight,

```

```

        on_delete=models.PROTECT,
        verbose_name="Рост",
    )
size = models.ForeignKey(
    ProductSize,
    on_delete=models.PROTECT,
    verbose_name="Размер",
)
count = models.IntegerField(
    default=0,
    verbose_name="Количество",
)

class Meta:
    verbose_name = "Особенность характеристики к товару"
    verbose_name_plural = "Особенности характеристики к товарам"

def __str__(self):
    return f'Размер: {str(self.size)}, Рост: {str(self.height)}'

class PictureProduct(models.Model):
    """
    Модель изображений продукта
    """

    image = models.FileField('Изображение', default=None)
    product = models.ForeignKey(Product, on_delete=models.CASCADE, null=True)

    class Meta:
        verbose_name = "Изображение"
        verbose_name_plural = "Изображения"

    def __str__(self):
        return f'Изображение к товару {self.id}'

from rest_framework import serializers

from product.models import (
    Category,
    Subcategory,
    Product,
    Brand,
    PictureProduct,
    ProductFeatureItem,
    ProductFeature,
)
class SubcategorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Subcategory
        fields = ('id', 'name')

```

```

class CategorySerializer(serializers.ModelSerializer):
    subcategories = serializers.SerializerMethodField()

    class Meta:
        model = Category
        fields = (
            'id',
            'name',
            'image',
            'subcategories',
        )

    def get_subcategories(self, obj):
        subcategories = Subcategory.objects.filter(category=obj)
        return SubcategorySerializer(subcategories, many=True).data


class BrandSerializer(serializers.ModelSerializer):

    class Meta:
        model = Brand
        fields = (
            'id',
            'name',
            'image',
        )


class ProductShortSerializer(serializers.ModelSerializer):
    brand = BrandSerializer()

    class Meta:
        model = Product
        fields = (
            'id',
            'is_new',
            'name',
            'model',
            'price',
            'brand',
            'image',
        )


class ProductPictureSerializer(serializers.ModelSerializer):
    class Meta:
        model = PictureProduct
        fields = ('image', )


class ProductSerializer(serializers.ModelSerializer):

```

```

brand = BrandSerializer()
images = serializers.SerializerMethodField('images_list')
features = serializers.SerializerMethodField('features_list')

class Meta:
    model = Product
    fields = (
        'id',
        'is_new',
        'name',
        'model',
        'price',
        'brand',
        'image',
        'images',
        'features',
    )
)

def images_list(self, obj):
    images_ = PictureProduct.objects.filter(product=obj)
    return ProductPictureSerializer(images_, many=True).data

def features_list(self, obj):
    items = ProductFeatureItem.objects.filter(product_feature__product=obj)
    return {
        'sizes': list(set([x.size.value for x in items.order_by('size_value')])),
        'heights': list(set([x.height.value for x in items.order_by('height_value')])),
        'colors': [{ 'name': x.color.name, 'value': x.color.value} for x in ProductFeature.objects.filter(product=obj)],
    }

import re
import os
import random
import string

from django.core.management.base import BaseCommand, CommandError
from django.core.files import File

from product.models import Product, Subcategory, Brand

IMAGE_EXTENSIONS = ('.jpg', '.png')
TITLES_LIST = (
    "Брюки жен.",
    "Комплект жен.",
    "Майка жен.",
    "Джемпер жен.",
)
PRICE_INTERVAL = (50, 200)

```

```

class Command(BaseCommand):

    def add_arguments(self, parser):
        parser.add_argument('--n', type=int)
        parser.add_argument('--path_images', type=str)

    def handle(self, *args, **options):
        path_images = options.get('path_images')
        if not os.path.exists(path_images):
            raise CommandError("Unknown image directory")

        count = options.get('n')
        if count <= 0 or count > 1000:
            raise CommandError("The number of items must be positive, but not
higher than 1000")

        images = [f_ for f_ in os.listdir(path_images) if
self.is_img_file(f_)]
        if not images:
            raise CommandError("Image directory is empty")

        data_images = self.remake_images_list(images)
        parent_dir = os.path.abspath(path_images)
        self.create_data_(data_images, count, parent_dir)

    @staticmethod
    def remake_images_list(images):
        pattern = re.compile(r'^(\d)_')
        result = {}
        for image in images:
            subcategory = pattern.findall(image.lower())

            if subcategory:
                subcategory = subcategory[0]
                if subcategory not in result.keys():
                    result[subcategory] = [image, ]
                continue
            result[subcategory].append(image)

        return result

    @staticmethod
    def create_data_(data, count, parent_path):
        subcategories_keys = list(data.keys())
        subcategories = Subcategory.objects.all()
        if subcategories.filter(id__in=subcategories_keys).count() !=
len(subcategories_keys):
            raise CommandError("Subcategory not found")

        brands = Brand.objects.all()

```

```

    for item in range(count):
        subcategory = Subcategory.objects.get(
            id=int(random.choice(subcategories_keys)),
        )
        model = ''.join(random.choices(string.ascii_uppercase +
string.digits, k=7))

        file_name = random.choice(data.get(str(subcategory.id)))
        file_path = os.path.join(parent_path, file_name)
        file_ = File(open(file_path, 'rb'))

        p = Product.objects.create(
            name=random.choice(TITLES_LIST),
            subcategory=subcategory,
            model=model,
            is_new=bool(random.getrandbits(1)),
            brand=random.choice(brands),
            price=random.randint(*PRICE_INTERVAL),
            image=File(file=open(file_path, 'rb'), name=file_name)
        )

    @staticmethod
    def is_img_file(path):
        _, ext = os.path.splitext(path)
        return ext.lower() in IMAGE_EXTENSIONS

from rest_framework import serializers

from reference.models import (
    ProductColor,
    ProductHeight,
    ProductSize,
)

class ProductColorSerializer(serializers.ModelSerializer):
    class Meta:
        model = ProductColor
        fields = ('id', 'name', 'value')

class ProductHeightSerializer(serializers.ModelSerializer):
    class Meta:
        model = ProductHeight
        fields = ('id', 'value')

class ProductSizeSerializer(serializers.ModelSerializer):
    class Meta:
        model = ProductSize
        fields = ('id', 'value')

from django.db import models

```

```

class ProductSize(models.Model):
    value = models.IntegerField(
        unique=True,
        verbose_name="Значение размера",
    )

    class Meta:
        verbose_name = "Размер"
        verbose_name_plural = "Размеры"

    def __str__(self):
        return str(self.value)

class ProductHeight(models.Model):
    value = models.IntegerField(
        unique=True,
        verbose_name="Значение высоты",
    )

    class Meta:
        verbose_name = "Высота"
        verbose_name_plural = "Высота"

    def __str__(self):
        return str(self.value)

class ProductColor(models.Model):
    name = models.CharField(
        max_length=32,
        verbose_name="Название цвета",
    )
    value = models.CharField(
        max_length=6,
        verbose_name="Цвет в формате HEX (000fff)",
        default=None,
        null=True,
    )

    class Meta:
        verbose_name = "Цвет"
        verbose_name_plural = "Цвета"

    def __str__(self):
        return self.name

"""

Django settings for svitanak project.

```

```
Generated by 'django-admin startproject' using Django 3.1.4.
```

```
For more information on this file, see  
https://docs.djangoproject.com/en/3.1/topics/settings/
```

```
For the full list of settings and their values, see  
https://docs.djangoproject.com/en/3.1/ref/settings/  
"""
```

```
import os  
from pathlib import Path  
  
# Build paths inside the project like this: BASE_DIR / 'subdir'.  
BASE_DIR = Path(__file__).resolve().parent.parent  
  
# Quick-start development settings - unsuitable for production  
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/  
  
# SECURITY WARNING: keep the secret key used in production secret!  
SECRET_KEY = '0lf0dcrtqydwidm0n&34x7i9!6vw$kk6rw@is*$hafx8!j$6o'  
  
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = True  
  
ALLOWED_HOSTS = ['svitanak-shop.site', '*']  
  
# Application definition  
  
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
  
    # Приложения  
    'reference',  
    'product',  
    'accounts.apps.AccountsConfig',  
  
    # Внешние зависимости  
    'corsheaders',  
    'django_extensions',  
    'nested_admin',  
  
    'rest_framework',  
    'rest_framework.authtoken',  
    'drf_registration',  
]  
  
MIDDLEWARE = [
```

```

'django.middleware.security.SecurityMiddleware',
'django.contrib.sessions.middleware.SessionMiddleware',
'corsheaders.middleware.CorsMiddleware',
'django.middleware.common.CommonMiddleware',
'django.middleware.csrf.CsrfViewMiddleware',
'django.contrib.auth.middleware.AuthenticationMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',
'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'svitanak.urls'

TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
},
]

WSGI_APPLICATION = 'svitanak.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {

    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'svitanak',
        'USER': 'svitanak',
        'PASSWORD': 'svitanak',
        'HOST': 'localhost',
        'PORT': 5432,
    }
}

# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
{
    'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',

```

```

        },
        {
            'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
        },
        {
            'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
        },
        {
            'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
        },
    ],
}

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
}

AUTHENTICATION_BACKENDS = [
    # 'drf_registration.auth.MultiFieldsModelBackend',
    'accounts.backends.EmailBackend',
]

AUTH_USER_MODEL = 'accounts.CustomUser'

PAGE_SIZE = 32

# Internationalization
# https://docs.djangoproject.com/en/3.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/


STATIC_URL = '/staticfiles/'
STATIC_ROOT = BASE_DIR / 'static'

MEDIA_URL = '/mediafiles/'
MEDIA_ROOT = BASE_DIR / 'media'

```

```

CORS_ORIGIN_ALLOW_ALL = True

EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'svitanoksvitankov@gmail.com'
EMAIL_HOST_PASSWORD = '5Gf1So0XCxmHNhqD'

EMAILS_FOR_MESSAGES = ['artem.nester.m@gmail.com', ]

DRF_REGISTRATION = {
    'USER_FIELDS': (
        'id',
        'email',
        'password',
    ),
    'USER_SERIALIZER': 'accounts.serializers.UserSerializer',
    'USER_WRITE_ONLY_FIELDS': (
        'password',
    ),
    'LOGIN_USERNAME_FIELDS': ['email'],
    'LOGOUT_REMOVE_TOKEN': True,
}

from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

admin.autodiscover()
admin.site.enable_nav_sidebar = False

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),

    path('_nested_admin/', include('nested_admin.urls')),
]

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

# Generated by Django 3.1.4 on 2020-12-09 22:19

from django.db import migrations, models


class Migration(migrations.Migration):

    initial = True

    dependencies = [
]

```

```

operations = [
    migrations.CreateModel(
        name='ProductCollection',
        fields=[
            ('id', models.AutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
            ('name', models.CharField(max_length=64)),
        ],
    ),
    migrations.CreateModel(
        name='ProductColor',
        fields=[
            ('id', models.AutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
            ('name', models.CharField(max_length=32)),
            ('color_hex', models.CharField(max_length=6)),
        ],
    ),
    migrations.CreateModel(
        name='ProductHeight',
        fields=[
            ('id', models.AutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
            ('value', models.IntegerField(unique=True)),
        ],
    ),
    migrations.CreateModel(
        name='ProductSize',
        fields=[
            ('id', models.AutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
            ('value', models.IntegerField(unique=True)),
        ],
    ),
]

```

Клиентская часть программного средства:

```
{
  "name": "svitanak",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "nuxt",
    "build": "nuxt build",
    "start": "nuxt start",
    "generate": "nuxt generate"
  },
  "dependencies": {
    "@nuxtjs/auth-next": "^5.0.0-1610115973.9fdaa66",
    "@nuxtjs/axios": "5.12.5",
  }
}
```

```

    "@nuxtjs/dotenv": "^1.4.1",
    "core-js": "^3.6.5",
    "nuxt": "^2.14.6",
    "v-mask": "^2.2.3"
},
"devDependencies": {}
}

export default {
// Disable server-side rendering (https://go.nuxtjs.dev/ssr-mode)
ssr: false,

// Global page headers (https://go.nuxtjs.dev/config-head)
head: {
  title: 'svitanak',
  meta: [
    {charset: 'utf-8'},
    {name: 'viewport', content: 'width=device-width, initial-scale=1'},
    {hid: 'description', name: 'description', content: ''}
  ],
  link: [
    {rel: 'icon', type: 'image/x-icon', href: '/favicon.ico'}
  ],
  script: []
},
static: {
  prefix: false
},
// Global CSS (https://go.nuxtjs.dev/config-css)
css: [
  '@/assets/css/libs.min.css',
  '@/assets/css/main.css',
],
// Plugins to run before rendering page (https://go.nuxtjs.dev/config-plugins)
plugins: [
  {src: '~plugins/vmask.js', ssr: false}
],
// Auto import components (https://go.nuxtjs.dev/config-components)
components: true,
// Modules for dev and build (recommended) (https://go.nuxtjs.dev/config-modules)
buildModules: ['@nuxtjs/dotenv'],
// Modules (https://go.nuxtjs.dev/config-modules)
modules: ['@nuxtjs/axios', '@nuxtjs/auth-next'],
auth: {

```

```

        strategies: {
          local: {

        }
      }
    },
  }

  // Build Configuration (https://go.nuxtjs.dev/config-build)
  build: {},
}

<template>
  <div>
    <!--[if lt IE 9]>
    <div class="browserupgrade"><p>You are using an <strong>outdated</strong>
    browser. Please <a
      href="http://browsehappy.com/">upgrade your browser</a> to improve your
    experience.</p></div> <![endif]-->
    <Header/>
    <div class="wrapper">
      <div class="max-wrap layout">
        <Nuxt/>

        </div>
      </div>
      <Sidebar/>
      <SearchWindow/>
      <CardWindow/>
      <MessageWindow/>
      <AuthWindow/>
      <LoginWindow/>
      <LeftMenu/>
      <Footer/>
    </div>
  </template>

<script>
  export default {
    mounted() {
      this.setViewProduct()
      this.getInitialData()
    },
    methods: {
      getInitialData: function () {
        const path = process.env.PATH_API_INITIAL + "?only=brands,co-
        lors,height,sizes"
        this.$axios.$get(path).then(
          (response) => {
            this.$store.commit('setCategories', response.categories)
            this.$store.commit('setBrandsFilter', response.brands)
            this.$store.commit('setColorsFilter', response.colors)

```

```

        this.$store.commit('setHeightsFilter', response.height)
        this.$store.commit('setSizesFilter', response.sizes)
        this.$store.commit('setOrdersByList', response.orders)
    }
)
},
setViewProduct: function () {
    const name = 'setProductBigView'
    if (process.browser) {
        const value = JSON.parse(localStorage.getItem(name))
        this.$store.commit(name, value)
    }
}
}
}
</script>

<template>
<div
    class="p-filters-tags__item p-filters-tags-item-js"
    data-filter-group="type"
>
    <i @click="delFromChosedFilter()">X</i>
    <span class="p-filters-tag-text-js">{{ getItemName() }}</span>
</div>
</template>

<script>
export default {
    props: ['item'],
    methods: {
        getItemName() {
            if (this.item.value.value != null && this.item.value.name != null) {
                return this.item.value.name
            } else if (this.item.value.value != null) {
                return this.item.value.value
            }
            return this.item.value.name
        },
        delFromChosedFilter() {
            this.$store.commit('delFromChosedFilter', {
                name: this.item.name,
                index: this.item.index,
                id: this.item.value.id
            })
        }
    }
}
</script>
```

```

<template>
  <div
    :class="[
      'p-filters-item',
      isOpen ? 'p-filters-is-open' : '',
      isActive ? 'p-filters-on' : ''
    ]"
  >
    <div class="p-filters-placeholder">Отметте необходимое значение
    </div>
    <div class="p-filters-selected"
      data-filter-value-prefix="Выбрано" data-filter-value-post-
      fix=""></div>
    <div class="p-filters-inner">
      <div class="p-filters-select" @click="changeViewBlock()">
        <div class="p-filters-select-head">
          <div class="p-filters__title">{{ title }}</div>
        </div>
        <div class="p-filters-angle"><i>&ampnbsp</i></div>
      </div>
      <div class="p-filters-drop">
        <ul class="p-filters-drop-list">
          <li
            v-for="(item, index) in list"
            :key="index"
            @click="test(index, item.id)"
          >
            <label onclick="return false;" class="check-label">
              <input onclick="return false;" type="checkbox"
                :checked="item.isActive" />
              <i>&ampnbsp</i>
              <span>{{ getNameByKey(item) }}</span>
            </label>
          </li>
        </ul>
      </div>
    </div>
  </template>
  <script>
    export default {
      props: [
        'list',
        'title',
        'key_name',
        'id',
        'openDefault',
        'storeName',
      ],
      data() {
        return {
          isOpen: false,

```

```

        isActive: false,
    }
},
mounted() {
    this.isOpenBlock()
},
methods: {
    changeViewBlock: function () {
        let name = 'block__' + this.id
        this.isOpen = !this.isOpen
        if (process.browser) {
            localStorage.setItem(name, this.isOpen)
        }
    },
    isOpenBlock: function () {
        let name = 'block__' + this.id
        if (process.browser) {
            let status = localStorage.getItem(name)
            if (status != null) {
                this.isOpen = JSON.parse(status)
            } else {
                this.isOpen = this.openDefault
            }
        }
    },
    getNameByKey: function (item) {
        return item[this.key_name]
    },
    test: function (index, id) {
        console.log(index, id)
        this.$store.commit('setItemFilterActiveValue', {name: this.storeName, index})
        this.$store.commit('addToChosedFilter', {name: this.storeName, index, id})
    },
    watch: {
        list: {
            deep: true,
            handler(value) {
                let status = false
                value.forEach(x => {
                    if (x.isActive) status = true;
                });
                this.isActive = status
            }
        }
    }
}

}
</script>

module.exports = {

```

```

apps: [
  {
    name: 'svitanak',
    exec_mode: 'cluster',
    instances: 'max',
    script: './node_modules/nuxt/bin/nuxt.js',
    args: 'start'
  }
]

<template>
<div class="m-aside m-aside-shutter-js">
  <div class="m-aside-layout">
    <form name="_form">
      <div class="m-aside-align">
        <div class="m-aside-holder">
          <div class="m-wrapper">
            <div class="p-filters-list">
              <FilterComponent
                :list="filtersData['brands']"
                title="Торговая марка"
                key_name="name"
                :id="1"
                :openDefault="false"
                storeName="brandsFilter"
              />
              <FilterComponent
                :list="filtersData['sizes']"
                title="Размер"
                key_name="value"
                :id="2"
                :openDefault="false"
                storeName="sizesFilter"
              />
              <FilterComponent
                :list="filtersData['heights']"
                title="Пост"
                key_name="value"
                :id="3"
                :openDefault="false"
                storeName="heightsFilter"
              />
              <FilterComponent
                :list="filtersData['new']"
                title="Новинка"
                key_name="name"
                :id="4"
                :openDefault="true"
                storeName="newFilter"
              />
              <FilterComponent
                :list="filtersData['colors']"
              />
            </div>
          </div>
        </div>
      </form>
    </div>
  </div>
</div>

```

```

        title="Цвет"
        key_name="name"
        :id="5"
        :openDefault="true"
        storeName="colorsFilter"
      />
    </div>
  </div>
</div>
<div class="m-aside-footer">
  <button onclick="return false" @click="resetFilters()" class="btn-underline btn-with-icon btn-clear-filters" :disabled="!isFiltersChose">
    <svg class="svg-ico-close" xmlns="http://www.w3.org/2000/svg" width="32" height="32">
      viewBox="0 0 57.2 57.2">
      <path
        d="M34.3 28.6L56 6.9c1.6-1.6 1.6-4.1 0-5.7 -1.6-1.6-4.1-1.6-5.7 0L28.6 22.9 6.9 1.3c-1.6-1.6-4.1-1.6-5.7 0 -1.6 1.6-1.6 4.1 0 5.7l21.7 21.6L1.3 50.3c-1.6 1.5-1.6 4.1 0 5.6 0.8 0.8 1.8 1.2 2.8 1.2s2-0.4 2.8-1.2l21.7-21.6L50.3 56c0.8 0.8 1.8 1.2 2.8 1.2s2-0.4 2.8-1.2c1.6-1.6 1.6-4.1 0-5.7L34.3 28.6z"></path>
    </svg>
    <span class="for-desk">Очистить фильтр</span>
    <span class="for-mob">Очистить</span>
  </button>
</div>
</form>
</div>
</div>
</template>

<script>
  import {mapGetters} from 'vuex'
  export default {
    props: ['filtersData'],

    methods: {
      resetFilters() {
        this.$store.commit('resetFilters')
      }
    },
    computed: {
      ...mapGetters([
        'isFiltersChose',
      ])
    }
  }
</script>

<template>
  <div class="p-options">

```

```

<div class="p-options__list">
  <div class="p-options__item">
    <div class="p-options__inner">
      <div class="p-options__group sorting-js order-asc">
        <ul class="p-thumbs">
          <li>
            <div class="select">
              <select @change="onChange($event)" name="sorting">
                <option value="0">Сортировка по:</option>
                <option
                  :value="id"
                  v-for="(value, id, index) in $store.state.orders"
                  :key="index"
                >
                  {{ value }}
                </option>
              </select>
            </div>
          </li>
        </ul>
      </div>
    </div>
  </div>
  <div class="p-options__item p-options__item--view-switcher">
    <div class="p-options__inner">
      <!--view switcher-->
      <div class="view-switcher"
        data-toggle-view-switcher="products">
        <a
          :class="['grid-view', !$store.state.bigProductView ? 'active' :
          '' ]"
          title="Сокращенный вид"
          @click="changeViewProduct(false)"
        >
          <svg class="svg-ico-menu-short"
            xmlns="http://www.w3.org/2000/svg" width="32"
            height="32" viewBox="0 0 177 83.7">
            <path
              d="M0 37.1c0.3 0 0.6 0.1 1 0.1 11.7 0 23.4 0 35.1 0 0.4 0
              0.7-0.1 1.2-0.1v-1.7c0-11.2 0-22.5 0-33.7 0-0.6 0-1.1 0.1-1.7H0V37.1zM139.6
              0c0 0.6 0.1 1.1 0.1 1.7 0 11.2 0 22.5 0 33.7v1.8h1.7c11.4 0 22.8 0 34.3 0 0.5
              0 0.9-0.1 1.4-0.1v0H139.6zM37.3 48.4v-1.8h-1.7c-11.4 0-22.8 0-34.2 0 -0.5 0-
              0.9 0.1-1.4 0.1v37.1h37.4c0-0.6-0.1-1.1-0.1-1.7C37.3 70.8 37.3 59.6 37.3
              48.4zM140.9 46.5c-0.4 0-0.7 0.1-1.2 0.1v1.7c0 11.2 0 22.5 0 33.7 0 0.6 0 1.1-
              0.1 1.7H177V46.6c-0.3 0-0.6-0.1-1-0.1c164.3 46.5 152.6 46.5 140.9 46.5zM83.7
              0H46.6c0 12.2 0 24.3 0 36.5 0 0.2 0.1 0.4 0.1 0.6h37c0-0.5 0.1-0.9 0.1-1.3 0-
              11.6 0-23.1 0-34.7C83.8 0.7 83.8 0.4 83.7 0zM93.3 37.1h37c0.1-0.2 0.1-0.3
              0.1-0.5 0-12.2 0-24.4 0-36.6H93.3c0 0.3-0.1 0.6-0.1 0.8 0 11.7 0 23.5 0
              35.2C93.2 36.4 93.3 36.8 93.3 37.1zM83.7 46.6H46.7c-0.1 0.2-0.1 0.3-0.1 0.5 0
              12.2 0 24.4 0 36.6h37.1c0-0.3 0.1-0.6 0.1-0.8 0-11.7 0-23.5 0-35.2C83.8 47.3
              83.7 47 83.7 46.6zM93.3 46.6c0 0.5-0.1 0.9-0.1 1.3 0 11.6 0 23.1 0 34.7 0 0.4
            </path>
          </svg>
        </a>
      </div>
    </div>
  </div>

```

```

0.1 0.7 0.1 1.1h37.1c0-12.2 0-24.3 0-36.5 0-0.2-0.1-0.4-0.1-
0.6H93.3z"></path>
        </svg>
        <span>Сокращенный вид</span>
    </a>
    <a
        :class="['list-view', $store.state.bigProductView ? 'active' :
        '' ]"
        title="Подробный вид"
        @click="changeViewProduct(true)"
    >
        <svg class="svg-ico-menu-full"
        xmlns="http://www.w3.org/2000/svg" width="32"
            height="32" viewBox="0 0 131 62">
            <path
                d="M60.4 27.5h1.6c0-0.4 0.1-0.7 0.1-1 0-8.6 0-17.2 0-25.9
0-0.2 0-0.4-0.1-0.6H0v27.4c0.5 0 0.9 0.1 1.4 0.1C21.1 27.5 40.7 27.5 60.4
27.5zM69 0c0 2.8-0.1 5.5-0.1 8.3 0 5.9 0 11.8 0 17.7v1.5H70.4c19.8 0 39.6 0
59.4 0 0.4 0 0.8 0 1.2-0.1V0H69zM1.2 34.4c-0.4 0-0.8 0-1.2 0.1v27.4h62c0-2.8
0.1-5.5 0.1-8.3 0-5.9 0-11.8 0-17.7v-1.5H60.6C40.8 34.4 21 34.4 1.2
34.4zM70.6 34.4h-1.6c0 0.4-0.1 0.7-0.1 1 0 8.6 0 17.2 0 25.9 0 0.2 0 0.4 0.1
0.6H131v-27.4c-0.5 0-0.9-0.1-1.4-0.1C109.9 34.4 90.3 34.4 70.6 34.4z"></path>
        </svg>
        <span>Подробный вид</span>
    </a>
</div>
<!--view switcher end-->
</div>
</div>
</div>
</div>
</template>

<script>
export default {
    methods: {
        changeViewProduct: function (value) {
            const name = 'setProductBigView'
            this.$store.commit(name, value)
            if (process.browser) {
                localStorage.setItem(name, value)
            }
        },
        onChange(event) {
            let value = event.target.value
            if (value === '0') {
                value = null
            }
            this.$store.commit('chooseOrderBy', value)
        },
    },
}
</script>

```

```

<style>
  .view-switcher a {
    cursor: pointer
  }
</style>

<template>
  <div class="pagination max-wrap-text">
    <div class="pg-left">
      <a
        v-if="pages.links.has_prev"
        class="pg-prev"
        @click="toPrevPage()"
      >
        <span>Назад</span>
        <i class="pg-prev-arr">&ampnbsp</i>
      </a>
      <a class="pg-prev disabled" v-else>
        <span>Назад</span>
        <i class="pg-prev-arr">&ampnbsp</i>
      </a>
    </div>
    <ul class="pg-list">
      <!--      <li><a href="#">1</a></li>-->
      <!--      <li><span class="dots">...</span></li>-->
      <!--      <li><a href="#">8</a></li>-->
      <li class="active"><a>{{ pages.current_page }}</a></li>
      <!--      <li><a href="#">10</a></li>-->
      <!--      <li><span class="dots">...</span></li>-->
      <!--      <li><a href="#">127</a></li>-->
    </ul>
    <div class="pg-right">
      <a
        v-if="pages.links.has_next"
        class="pg-next"
        @click="toNextPage"
        style="cursor:pointer;"
      >
        <span>Вперед</span>
        <i class="pg-next-arr">&ampnbsp</i>
      </a>
      <a v-else class="pg-next disabled">
        <span>Вперед</span>
        <i class="pg-next-arr">&ampnbsp</i>
      </a>
    </div>
    {{ pages }}
  </div>
</template>

<script>

```

```

export default {
  props: ['pages'],
  data() {
    return {}
  },
  mounted() {
    console.log()
  },
  methods: {
    toNextPage() {
      let page = null
      if (this.$store.state.page != null) {
        page = this.$store.state.page
      } else {
        page = 1
      }
      this.$router.push({name: 'index', query: {page: page + 1}})
      this.$store.commit('setPage', page + 1)
    },
    toPrevPage() {
      let page = null
      if (this.$store.state.page != null) {
        page = this.$store.state.page
      } else {
        page = 1
      }
      this.$router.push({name: 'index', query: {page: page - 1}})
      this.$store.commit('setPage', page - 1)
    },
  },
  computed: {}
}
</script>

<template>
  <article class="products__item">
    <div class="products__inner">
      <div class="products__brand">
        
      </div>
      <NuxtLink :to="{name: 'products-id', 'params': {id: data.id}}">
        <div class="products__visual">
          <div v-if="data.is_new" class="products__new">&ampnbsp</div>
          <div class="products__figure">
            <div class="products__img">
              
            </div>
          </div>
        </div>
        <div class="products__title"><span>{{ data.name }}</span> <em
          class="products__model">{{ data.model }}</em></div>
      </NuxtLink>
      <div class="products__content">

```

```

<div class="products__dl">
    <div class="products__di">
        <div class="products__dt">Модель:</div>
        <div class="products__dd">{{ data.model }}</div>
    </div>
    <div class="products__di">
        <div class="products__dt">Коллекция:</div>
        <div class="products__dd"><a href="#">Лето</a></div>
    </div>
    <div class="products__di">
        <div class="products__dt">Полотно:</div>
        <div class="products__dd">Кулир</div>
    </div>
    <div class="products__di">
        <div class="products__dt">Состав:</div>
        <div class="products__dd">Хлопок 100%</div>
    </div>
    <div class="products__di">
        <div class="products__dt">Размеры:</div>
        <div class="products__dd">84-108</div>
    </div>
    <div class="products__di">
        <div class="products__dt">Цвет</div>
        <div class="products__dd">
            <div class="color-pics">
                <div class="color-pic" style="background-color: #a9e2f5;">#a9e2f5</div>
                <div class="color-pic" style="background-color: #f2d7fa;">#f2d7fa</div>
            </div>
        </div>
    </div>
    <div class="products__footer">
        <div class="products__price">
            <div class="cur"><strong class="val">{{ data.price }}</strong>
<span class="unit">BYN</span>
        </div>
    </div>
    <div class="products__basket">
        <a href="product-card.html#order-1" class="btn-to-basket" title="Добавить в корзину">
            <svg class="svg-ico-basket-add" xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 40.6 29.3">
                <path d="M40.6 27.8c-0.9-7-1.7-13.2-1.2-19.7 0-0.4-0.1-0.7-0.4-1s-0.6-0.4-1-0.4h-4.2v-0.9c0.1-3.2-2.3-5.8-5.2-5.8 -2.9 0-5.3 2.6-5.3 5.8v0.9h-4.2c-0.7 0-1.3 0.6-1.3 1.3v2.2c0 5.8 0 11.8-1.2 17.5 -0.1 0.4 0 0.8 0.3 1.1 0.3 0.3 0.6 0.5 1 0.5h21.4c0.4 0 0.8-0.2 1-0.5C40.6 28.6 40.7 28.2 40.6 27.8zM25.9 5.8c0-1.7 1.2-3.2 2.7-3.2s2.7 1.4 2.7 3.2v0.8h-5.4V5.8zM19.5 26.7c1-5.5 0.9-11 0.9-16.4v-0.9h2.9v4.7h2.6V9.4h5.3v4.7h2.6V9.4h2.8c-0.3 5.8

```

```

0.3 11.3 1.1 17.3H19.5zM12.1 16.7c-1-1.1-2.1-3.3-3.1 -0.3-0.2-0.6-0.5-
0.9-0.6H7.5c-0.2 0.1-0.4 0.1-0.6 0.3 -0.6 0.5-0.6 1.3 0 1.8 0.5 0.4 1 0.9 1.5
1.3l0.1 0.1H1.3c-0.5 0-1 0.3-1.2 0.8 0 0.1 0 0.2-0.1 0.2v0.3c0.2 0.7 0.6 1
1.3 1h7.1c-0.1 0.1-0.1 0.1-0.2 0.1 -0.5 0.5-1.1 0.9-1.6 1.4s-0.4 1.4 0.2
1.8c0.1 0.1 0.3 0.1 0.5 0.2h0.4c8 22.2 8.3 22 8.6 21.8c1.2-1 2.3-2 3.5-3.1
0.3-0.3 0.6-0.5 0.6-0.9v-0.3c12.6 17.2 12.4 16.9 12.1 16.7z"></path>
</svg>
<span>В корзину</span>
</a>
<a href="#" class="btn-from-basket" title="Удалить из корзины">
<svg class="svg-ico-basket" xmlns="http://www.w3.org/2000/svg"
width="24"
height="24" viewBox="0 0 24 29.3">
<path
d="M24 27.8c-0.9-7-1.7-13.2-1.2-19.7 0-0.4-0.1-0.7-0.4-1 -
0.3-0.3-0.6-0.4-1-0.4h-4.2V5.8c17.3 2.6 14.9 0 12 0S6.7 2.6 6.7
5.8v0.9H2.5C1.8 6.7 1.2 7.3 1.2 8v2.2c1.2 16 1.2 22 0 27.7c-0.1 0.4 0 0.8 0.3
1.1s0.6 0.5 1 0.5h21.4c0.4 0 0.8-0.2 1-0.5c24 28.6 24.1 28.2 24 27.8zM9.3
5.8c0-1.7 1.2-3.2 2.7-3.2s2.7 1.4 2.7 3.2v0.8H9.3V5.8zM2.9 26.7c1-5.5 0.9-11
0.9-16.4V9.4h2.9v4.7h2.6V9.4h5.3v4.7h2.6V9.4H20c-0.3 5.8 0.3 11.3 1.1
17.3H2.9z"></path>
</svg>
<span>Удалить из корзины</span>
</a>
</div>
</div>
</div>
</article>
</template>

<script>
export default {
  props: ['data'],
  methods: {
    getImgUrl: function (path) {
      return process.env.PATH_MEDIA + path
    }
  },
  computed: {
    getImgUrlProduct: function () {
      if (this.data) {
        return this.getImgUrl(this.data.image)
      }
    },
    getImgUrlBrand: function () {
      if (this.data) {
        return this.getImgUrl(this.data.brand.image)
      }
    },
  }
}
</script>

```

```

export const state = () => ({
  categories: [],
  brandsFilter: [],
  colorsFilter: [],
  heightsFilter: [],
  sizesFilter: [],
  newFilter: [
    {
      'id': true,
      'name': 'new',
      'isActive': false
    }
  ],
  choosedFilters: {
    'brandsFilter': {
      'values': [],
      'name': 'brands',
    },
    'colorsFilter': {
      'values': [],
      'name': 'colors',
    },
    'heightsFilter': {
      'values': [],
      'name': 'height',
    },
    'sizesFilter': {
      'values': [],
      'name': 'sizes',
    },
    'newFilter': {
      'values': [],
      'name': 'only_new',
    },
  },
  bigProductView: null,
  orders: [],
  orderBy: null,
  page: null,
  category: null,
})

export const mutations = {
  setPage(state, value) {
    this.state.page = value
  },
  setCategory(state, value) {
    this.state.category = value
  },
  setOrdersByList(state, data) {
    if (state.orders.length === 0) {

```

```

        state.orders = data
    }
},
chooseOrderBy(state, value) {
    state.orderBy = value
},
setCategories(state, data) {
    if (state.categories.length === 0) {
        state.categories = data
    }
},
setBrandsFilter(state, data) {
    if (state.brandsFilter.length === 0) {
        state.brandsFilter = data.map(v => ({...v, isActive: false}))
    }
},
setColorsFilter(state, data) {
    if (state.colorsFilter.length === 0) {
        state.colorsFilter = data.map(v => ({...v, isActive: false}))
    }
},
setHeightsFilter(state, data) {
    if (state.heightsFilter.length === 0) {
        state.heightsFilter = data.map(v => ({...v, isActive: false}))
    }
},
setSizesFilter(state, data) {
    if (state.sizesFilter.length === 0) {
        state.sizesFilter = data.map(v => ({...v, isActive: false}))
    }
},
setItemFilterActiveValue(state, {name, index}) {
    state[name][index].isActive = !state[name][index].isActive
},
addToChosedFilter(state, {name, index, id}) {
    if (state[name][index].isActive === true) {
        state.chosedFilters[name]['values'].push({index, name, value: state[name][index]})
    } else {
        state.chosedFilters[name]['values'].forEach(function (value, i) {
            if (value.value.id === id) {
                state.chosedFilters[name]['values'].splice(i, 1)
            }
        });
    }
},
delFromChosedFilter(state, {name, index, id}) {
    state[name][index].isActive = false
    state.chosedFilters[name]['values'].forEach(function (value, i) {
        if (value.value.id === id) {
            state.chosedFilters[name]['values'].splice(i, 1)
        }
    })
}

```

```

        });
    },
setProductBigView(state, value) {
    state.bigProductView = value
},
resetFilters(state) {
    let names = ['brandsFilter', 'colorsFilter', 'heightsFilter', 'sizesFilter', 'newFilter']
    names.forEach(function (name) {
        state[name].forEach(function (value) {
            value.isActive = false
        })
    })
}

for (const [key, value] of Object.entries(state.chosedFilters)) {
    state.chosedFilters[key]['values'].splice(0, state.chosedFilters[key]['values'].length)
}

},
}

export const getters = {
isFiltersChose: (state) => {
let status = false
for (const [key, value] of Object.entries(state.chosedFilters)) {
    if (value.values.length > 0) {
        status = true
        return status
    }
}
return status
},
getFilterParamsURL: (state) => {
let urls = []
for (const [key, value] of Object.entries(state.chosedFilters)) {
    let values = []
    if (value.values.length > 0) {
        value.values.forEach(function (v) {
            if (v.value.isActive) {
                values.push(v.value.id)
            }
        })
    }
    if (values.length > 0) {
        urls.push(value.name + '=' + values.join(','))
    }
}
return urls.join('&')
},
getParamsURL: (state) => {
let params = []
if (state.category != null) {

```

```

        params.push(`category=${state.category}`)
    }
    if (state.orderBy != null) {
        params.push(`order_by=${state.orderBy}`)
    }
    if (state.page != null) {
        params.push(`page=${state.page}`)
    }
    return params.join('&')
},
getOrderBy: (state) => {
    return state.orderBy
},
getPageValue: (state) => {
    return state.page
},
getCategoryValue: (state) => {
    return state.category
}
}

export const state = () => ({
    isOpen: false
})

export const mutations = {
    openSearch(state) {
        state.isOpen = !state.isOpen
    }
}

export const getters = {
    getOpenSearch: (state) => {
        return state.isOpen
    }
}

<template>
<div>
    <!--MAIN-->
    <div class="main">
        <div class="main__holder">
            <!--CONTENT-->
            <div class="content-wrap">
                <div class="content user-content">
                    <div class="breadcrumbs">
                        <ul class="breadcrumbs__list">
                            <li><span title="Главная">Главная</span></li>
                        </ul>
                    </div>
                    <div class="article-entry">

```

```

<h1>{{ categoryName ? categoryName : 'Продукция' }}</h1>
<div class="article-entry__holder text-slide-js" data-btn-text-
full="Текст полностью"
    data-btn-text-short="Свернуть">
    <p>В каталоге ОАО «Світанак» представлены актуальные на сего-
дняшний день модели белья и верхней
    одежду, выпускаемой нашим предприятием, а также вся необхо-
димая информация по каждой модели,
    включающая фото и номер модели, ассортиментную группу,
    шкалу размеров, состав, полотно и вид
    отделки.</p>
</div>
</div>
<div class="box-options">
    <div class="box-options__label">Для удобства пользования ката-
лог разбит на линии, а также снабжен
    удобным фильтром.
</div>
<div class="box-options__content">
    <!--departments-->
    <ul class="dep-list dep-list--horizontal equal-height-js">
        <li
            class="dep-item"
            v-for="(item, index) in $store.state.categories"
            :key="index"
        >
            <figure class="dep-img">
                <div>
:alt="item.name"></div>
                <div>
:alt="item.name"></div>
            </figure>
            <ul class="dep-sub-list">
                <li
                    v-for="(subitem, subindex) in item.subcategories"
                    :key="subindex"
                >
                    <a
                        style="cursor:pointer;"
                        :title="subitem.name"
                        @click="chooseCategory(subitem.id, subitem.name)"
                    >
                        <span>{{ subitem.name }}</span>
                    </a>
                </li>
            </ul>
        </li>
    </ul>
</div>
<!--Во время подбора результатов добавляем класс js-loading на
контейнер с классом p-filters-js-->
<div class="m-wrapper">
```

```

<div class="p-filters-tags">
  <div class="p-filters-tags__holder">
    <div class="p-filters-tags__frame">
      <span
        v-for="(value, key, index) in this.$store.state.chose-
Filters"
        :key="index"
      >
        <FilterChosedTop
          v-for="(item, index_) in value.values"
          :key="index_"
          :item="item"
        />
      </span>
    </div>
  </div>
</div>
<div class="m-container">
  <FiltersSidebar :filtersData="filters"/>
  <div class="m-content">
    <div class="m-content__holder">
      <FiltersTop/>
      <div class="note bottom-space-lg">Цены в белорусских руб-
лях приведены с учетом НДС
      </div>
      <div class="p-filters-tags">
        <div class="p-filters-tags__holder">
          <div class="p-filters-tags__frame p-filters-tags-js">
            </div>
        </div>
      </div>
      <div
        :class="[
          'products',
          !$store.state.bigProductView ? 'grid-view': '',
        ]"
        data-toggle-view-panels="products"
      >
        <section class="products__list">
          <ProductShort
            v-for="(item, index) in products_list"
            :key="index"
            :data="item"
            v-if="products_list.length > 0"
          />
          <div v-if="products_list.length == 0">
            Товары не найдены.
          </div>
        </section>
      </div>
      <Pagination v-if="products_list.length > 0" :pages="pagi-
nation"/>
    </div>
  </div>

```

```

        </div>
    </div>
    <div class="loader-cover">&nbsp;</div>
    </div>
    </div>
</div>
</div>

</div>
</template>

<script>
  import {mapGetters} from 'vuex'

  export default {
    head() {
      return {
        title: "Каталог"
      }
    },
    data() {
      return {
        filters: {
          'brands': [],
          'colors': [],
          'heights': [],
          'sizes': [],
          'new': this.$store.state.newFilter
        },
        products_list: [],
        pagination: {
          'links': {
            'has_next': false,
            'has_prev': false,
          },
          'total_pages': 1,
          'current_page': 1,
        },
        filtersURLParams: null,
        categoryName: null,
      }
    },
    mounted() {
      this.loadScript('js/common.js')
      this.getProductsList()

      if (this.checkQueryUrlIsInt(this.$route.query.category)) {
        this.$store.commit('setCategory', parseInt(this.$route.query.category))
      }
      if (this.checkQueryUrlIsInt(this.$route.query.page)) {
        this.$store.commit('setPage', parseInt(this.$route.query.page))
      }
    }
  }
</script>

```

```

        }
    }
    ,
    methods: {
        checkQueryUrlIsInt(value) {
            if (typeof value != "string") return false
            return !isNaN(value) && !isNaN(parseFloat(value))
        },
        getImgUrl: function (path) {
            return process.env.PATH_MEDIA + path
        },
        getProductsList() {
            let path = process.env.PATH_API_PRODUCTS
            let params = []
            if (this.getParamsURL !== '') {
                params.push(this.getParamsURL)
            }
            if (this.getFilterParamsURL !== '') {
                params.push(this.getFilterParamsURL)
            }
            if (params.length > 0) {
                path = path + '?' + params.join('&')
            }

            this.$axios.$get(path).then(
                (response) => {
                    this.products_list = response.results

                    this.pagination['links'] = response.links
                    this.pagination['total_pages'] = response.total_pages
                    this.pagination['current_page'] = response.current_page
                }
            ).catch((response) => {
                this.products_list = []
            });
        },
        loadScript(name) {
            if (!process.server) {
                var script = document.createElement("script");
                script.onload = this.onScriptLoaded;
                script.type = "text/javascript";
                script.src = name;
                document.body.appendChild(script);
            }
        },
        choseCategory(id, name) {
            this.categoryName = name
            this.$store.commit('setCategory', id)
            let query = {'category': id}
            if (this.checkQueryUrlIsInt(this.$route.query.page)) {
                query['page'] = this.$route.query.page
            }
        }
    }
}

```

```

        this.$router.push({ 'name': 'index', query: query })
    },
}

computed: {
    brandsFilter() {
        return this.$store.state.brandsFilter
    },
    colorsFilter() {
        return this.$store.state.colorsFilter
    },
    heightsFilter() {
        return this.$store.state.heightsFilter
    },
    sizesFilter() {
        return this.$store.state.sizesFilter
    },
    ...mapGetters([
        'getFilterParamsURL',
        'getOrderBy',
        'getPageValue',
        'getCategoryValue',
        'getParamsURL',
    ])
},
watch: {
    getPageValue(val) {
        this.getProductsList()
    },
    getCategoryValue() {
        this.getProductsList()
    },
    getFilterParamsURL(val) {
        this.getProductsList()
        this.filtersURLParams = val
    },
    getOrderBy(val) {
        this.getProductsList()
    },
    brandsFilter(new_value) {
        if (new_value.length > 0) {
            this.filters['brands'] = new_value
        }
    },
    colorsFilter(new_value) {
        if (new_value.length > 0) {
            this.filters['colors'] = new_value
        }
    },
    heightsFilter(new_value) {
    }
}
</script>

```