МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

Гомельский государственный технический университет имени

П.О. Сухого

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

специальность 1-40 04 01 «Информатика и технологии программирования»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к дипломной работе**

**на тему**

**«*Web*-приложение по учёту и мониторингу выполнения заказов фирмы по производству мебели»**

Разработал студент гр. ИП-41                        Гринь А.С.

(подпись) (Ф.И.О.)

Руководитель работы                        Мурашко В.С.

(подпись) (ученое звание, ученая степень, Ф.И.О.)

Консультант   
по экономической части                          доцент, к.э.н. Соловьева Л.Л.

(подпись) (ученое звание, ученая степень, Ф.И.О.)

Консультант по охране труда

и технике безопасности                        профессор, д.т.н. Кудин В.П.

(подпись) (ученое звание, ученая степень, Ф.И.О.)

Нормоконтроль                         Романькова Т.Л.

(подпись) (ученое звание, ученая степень, Ф.И.О.)

Рецензент

(подпись) (ученое звание, ученая степень, должность,

организация, Ф.И.О.)

Дипломная работа (\_\_\_\_\_\_стр.) допущена к защите   
в Государственной экзаменационной комиссии.

Зав. кафедрой   
«Информатика»                        доцент, к.т.н., Трохова Т.А.

(подпись) (ученое звание, ученая степень, Ф.И.О.)

Гомель 2024

**Лист с заданием**

**Реферат**

*WEB*-ПРИЛОЖЕНИЕ ПО УЧЁТУ И МОНИТОРИНГУ ВЫПОЛНЕНИЯ ЗАКАЗОВ ФИРМЫ ПО ПРОИЗВОДСТВУ МЕБЕЛИ: дипломная работа / А. С. Гринь. – Гомель: ГГТУ им. П. О. Сухого, 2024. – Дипломная работа: 112 страниц, 70 рисунков, 21 таблица, 20 источников, 3 приложения.

Ключевые слова: мебель, учёт заказов, склад, клиенты, технологии, мониторинг.

Объектом разработки является *web*-приложение по учёту и мониторингу выполнения заказов фирмы по производству мебели.

Цель работы: разработка и создание *web*-приложения по учёту и мониторингу выполнения заказов фирмы по производству мебели.

Характеристика проделанной работы: в ходе данной работы было разработано *web*-приложение по учёту и мониторингу выполнения заказов фирмы по производству мебели. Перед разработкой был проведен аналитический обзор, изучена предметная область и выявлена актуальность темы разрабатываемого приложения. В процессе разработки были пройдены этапы постановки задачи, проектирования, реализации и тестирования. Была разработана функциональная модель, структура и состав информационного обеспечения *web*-приложения. Особое внимание уделялось интуитивно понятному интерфейсу, удобству навигации и поиска необходимой информации. Кроме того, был выполнен экономический анализ проекта, подтвердивший его целесообразность. Таким образом, все поставленные цели разработки *web*-приложения были достигнуты.

Студент-дипломник подтверждает, что дипломная работа выполнена самостоятельно, приведенный в дипломной работе материал объективно отражает состояние разрабатываемого объекта, пояснительная записка проверена в системе «Атиплагиат»: *https://antiplagius.ru*. Процент оригинальности составляет 89%. Все заимствованные из литературных и других источников теоретические и методологические положения и концепции сопровождаются ссылками на источники, указанными в «Списке использованных источников».

**Резюме**

Тема работы: «*Web*-приложение по учёту и мониторингу выполнения заказов фирмы по производству мебели».

Объектом исследования являются наиболее эффективные и функциональные методы по учёту и мониторингу заказов.

Целью работы является разработка и создание *web*-приложения по учёту и мониторингу выполнения заказов фирмы по производству мебели, которое позволит клиентам осуществлять поиск мебели, заказывать мебель из каталога или с заданными параметрами, отслеживать свои заказы, оставлять отзывы, поможет сотрудникам вести учёт заказов и товаров, получать статистику по заказам.

Основным результатом работы является успешное выполнение поставленной задачи в полном объеме.

**Рэзюмэ**

Тэма дыпломнай работы: «*Web*-дадатак па ўліку і маніторынгу выканання заказаў фірмы па вытворчасці мэблі».

Аб'ектам даследавання з'яўляюцца найбольш эфектыўныя і функцыянальныя метады па ўліку і маніторынгу заказаў.

Мэтай работы з'яўляецца распрацоўка і стварэнне *web*-дадатка па ўліку і маніторынгу выканання заказаў фірмы па вытворчасці мэблі, якое дазволіць кліентам ажыццяўляць пошук мэблі, заказваць мэблю з каталога або з зададзенымі параметрамі, адсочваць свае замовы, пакідаць водгукі, дапаможа супрацоўнікам весці ўлік заказаў і тавараў, атрымліваць статыстыку па заказах.

Асноўным вынікам працы з'яўляецца паспяховае выкананне пастаўленай задачы ў поўным аб'ёме.

**Summary**

The topic of the work: «Web-application for accounting and monitoring the fulfillment of orders for a furniture manufacturing company».

The object of the study is the most effective and functional methods for accounting and monitoring orders.

The goal of the work is to develop and create a web application for accounting and monitoring the fulfillment of orders from a furniture manufacturing company, which will allow clients to search for furniture, order furniture from a catalog or with specified parameters, track their orders, leave reviews, and help employees keep track of orders and goods, receive statistics on orders.

The main result of the work is the successful completion of the assigned task in full.

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc169692200)

[1 Аналитический обзор существующих методов и средств автоматизации 7](#_Toc169692201)

[1.1 Анализ предметной области 7](#_Toc169692202)

[1.2 Обзор и анализ существующих аналогов 9](#_Toc169692203)

[1.3 Обзор технологий для реализации ПО 12](#_Toc169692204)

[1.4 Постановка задачи 15](#_Toc169692205)

[2 Проектирование программного обеспечения 17](#_Toc169692206)

[2.1 Функциональное моделирование 17](#_Toc169692207)

[2.2 Информационное обеспечение приложения 19](#_Toc169692208)

[2.3 Архитектура приложения 26](#_Toc169692209)

[3 Структура и реализация программного обеспечения 28](#_Toc169692210)

[3.1 Описание реализации программного комплекса 28](#_Toc169692211)

[3.2 Описание пользовательского интерфейса 31](#_Toc169692212)

[4 Тестирование программного обеспечения 49](#_Toc169692213)

[4.1 Ручное тестирование программного комплекса 49](#_Toc169692214)

[5 Экономическое обоснование дипломной работы 60](#_Toc169692215)

[5.1 Расчёт общей трудоёмкости разработки программного обеспечения 60](#_Toc169692216)

[5.2 Расчёт затрат на разработку программного продукта 64](#_Toc169692217)

[5.3 Формирование цены при создании программного обеспечения 73](#_Toc169692218)

[5.4 Расчёт эффекта от внедрения программного продукта 74](#_Toc169692219)

[6 Охрана труда и техника безопасности 77](#_Toc169692220)

[6.1 Государственное управление охраной труда 77](#_Toc169692221)

[6.2 Цели и задачи государственного управления охраны труда 78](#_Toc169692222)

[6.3 Органы государственного управления охраной труда и их полномочия 80](#_Toc169692223)

[6.4 Инструкции по охране труда 81](#_Toc169692224)

[7 Ресурсо- и энергосбережение при внедрении программного обеспечения 83](#_Toc169692225)

[7.1 Основные понятия в области ресурсо- и энергосбережения 83](#_Toc169692226)

[7.2 Ресурсо- и энергосбережения в результате внедрения разработанного ПО 84](#_Toc169692227)

[Заключение 86](#_Toc169692228)

[Список использованных источников 87](#_Toc169692229)

[Приложение А Листинг классов для работы с базой данных 89](#_Toc169692230)

[Приложение Б Листинг классов для организации взаимодействия пользователя с приложением 94](#_Toc169692231)

[Приложение В Листинг классов контроллеров 102](#_Toc169692232)

# **ВВЕДЕНИЕ**

В настоящее время компьютеров, мобильных средств связи, интернета и высоких технологий все более актуальным становится использование компьютерных систем, *web*-приложений для автоматизации бизнеса и работы специалиста. Предприятия мебельной отрасли давно осознали необходимость внедрения автоматизированных систем управления, учёта и контроля. Автоматизация заказов мебели позволяет оптимизировать все основные бизнес-процессы управления и повысить их эффективность. Обмен информацией, передача отчетов, оформление различных документов реализуется намного быстрее и удобнее. Все операции по обработке данных, поиску, выполнению запросов можно совершать дистанционно.

В связи с тенденцией роста интернет-торговли и увеличением количества клиентов, которые предпочитают покупать мебель онлайн, было принято решение разработать *web*-приложение.

*Web*-приложение – это программа с определенными функциями, которая работает через браузер. *Web*-приложение, в отличие от сайта, направлено на взаимодействие с клиентом, обработку данных и их хранение, в то время как сайт представляет собой ресурс для предоставления информации. В качестве примеров *web*-приложений можно выделить интернет-банкинг, интернет-магазины, системы бронирования (гостиниц, товаров, услуг) и так далее.

*Web*-приложение поможет автоматизировать работу компаний, занимающихся производством и продажей мебели. Данная программа позволит усовершенствовать процесс обслуживания клиентов благодаря ведению всестороннего учёта.

Целью данной работы является разработка *web*-приложения по учёту и мониторингу выполнения заказов фирмы по производству мебели на основе технологий *C#*, *ASP.NET Core MV*C. Приложение будет предоставлять широкий спектр функций, включая регистрацию пользователей, вход в систему, поиск и отображение информации о различной мебели, отслеживание заказов, возможность оставить отзыв на сайте, расчет статистики, уведомления и оповещения клиентов, управление складом и проверка его состояния.

1. **АНАЛИТИЧЕСКИЙ ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ И СРЕДСТВ АВТОМАТИЗАЦИИ**
2. **Анализ предметной области**

Оснащение своего дома мебелью и её правильная расстановка считается важной частью интерьера. Меблировка квартиры отражает вкусы хозяина и его успех в финансовом плане. Мебель занимает основное место в жизни человека из большого числа предметов и вещей, которые он применяет ежедневно. Из-за чего, очень трудно представить себе, как современный человек может обойтись без мебели. Мебельные товары занимают значительное место в жизни человека, оказывая влияние на физиологическое и психологическое его состояние и сопровождает человека на протяжении всей жизни.

Мебель окружает человека с давних времен. Многие историки и археологи убеждены, что первые предшественники современной мебели появились уже у первобытных людей.  Часто мебель рассматривалась в качестве необходимых и функциональных предметов помещения. Часто мебель ценилась за свою привлекательность и вычурность форм, а функциональному назначению уделялось не так много внимания. Люди прошлых времен имели свое собственное представление о мебели. Они использовали полые куски стволов дерева, чтобы складывать в них другие предметы. Так что можно сказать, что это было первое подобие шкафов. Более того, некоторые историки убеждены, что доисторические предки делали такие «шкафы» специально, выдалбливая в дереве ниши. Конечно, такие процессы нельзя отнести к мебельному производству. Хотя, если подумать, они являются именно созданием предметов интерьера. Просто выглядели они совсем иначе, чем те, к которым мы привыкли. Мебель отражает вкус, мировосприятие, предпочтения и достаток своего хозяина.

Форма мебели, её цвет и отделка показывают специфические национальные черты, географические характеристики и даже влияние эпохи. История мебели, улучшение её дизайна и форм представляют собой одну из тенденций в развитии мебели. Так как мебель необходима для удобства и уюта, при её создании большое место уделяют специальным задачам: мебель конструируется не только с учетом функциональности, но и с учетом красоты и эстетичного внешнего вида. Набор мебели, который состоит из самых обычных предметов, должен превращать жилое помещение в уютную и комфортную квартиру.

Мебель, которая изготавливается в производственных масштабных условиях, представляет собой одно из средств создания человеком своего микроклимата. Работа человека, которая направлена на создание интерьера, считается по-настоящему креативной и интересной. Мебели принадлежит огромное количество полезных и практических функций в каждом доме, квартире, офисе и других учреждениях. Мебельное производство находится в постоянном движении. Дизайн мебели постоянно изменяется и усовершенствуется, используются современные технические и конструкторские приемы и новые материалы для ее изготовления.

Совершенствуя сырьевые материалы и производственный цикл, совершен-

ствуется и мебель, становясь более комфортной, удобной в эксплуатации и эстетичной. По применяемому сырью мебельные материалы классифицируют:

* древесные;
* металлические;
* стеклянные;
* полимерные;
* текстильные,
* кожевенные и др.

Древесные материалы в производстве мебели древесные занимают основное место. Древесина многих пород имеет красивую поверхность (текстуру), а также отличается от других материалов легкостью обработки, соединения частей и отделки поверхности. Доски, бруски, древесностружечные плиты (ДСП), шпон, древесноволокнистые плиты (ДВП), мебельные щиты, клееная фанера являются основными видами древесных материалов. Мебель, изготовленная из стопроцентного древесного массива, экологически чистая, высокопрочная, функциональная и отличается самобытностью дизайна.

Мебельные предприятия специализируются на изготовлении мебели для помещений различного назначения, включая жилые, общественные и офисные. Большую часть в этой отрасли занимают предметы интерьера, предназначенные для дома. Производством мебели занимаются и крупные предприятия, и малый бизнес. Одни из них наладили выпуск корпусных изделий, другие производят диваны и кресла, но, принцип работы их мебельного производства идентичен. Согласно [1], производство мебели в Республике Беларусь осуществляют более 1200 организаций различных форм собственности.

Ассортимент выпускаемой мебели очень широк и разнообразен. Предприятия отрасли производят как простую мебель малых форм из недорогих материалов, так и высокохудожественные наборы и гарнитуры из натуральной древесины с использованием точения и резьбы. Выпускаются практически все виды корпусной и мягкой мебели: гостиные, спальни, кухни, детские, прихожие, мебель для ванных комнат, для загородных домов и агроусадеб, садовая мебель и мебель специального назначения.

Мебель, как продукция продолжительной эксплуатации, имеет довольно сложный производственный процесс. Процесс производства мебели может быть полностью автоматизированным или с использованием ручного труда. Второй вариант более характерен для малого бизнеса. На крупных предприятиях чаще используют автоматизированное оборудование, которое позволяет добиться высокой точности, снизить количество брака и уменьшить трудозатраты.

Главная задача *web*-приложения по учёту и мониторингу выполнения заказов фирмы по производству мебели – это обеспечить эффективное управление и мониторинг выполнения заказов, начиная с принятия заказа от клиента и заканчивая его завершением и доставкой, а также предоставление клиентам полного спектра услуг по оформлению заказов мебели.

Одной из основных функций *web*-приложения является регистрация клиентов и предоставление им информации о продаваемой мебели. Клиенты могут зарегистрироваться в системе, указав свои логин и пароль, или зайти в уже существующий аккаунт. Клиенты могут осуществлять поиск мебели по названию, либо смотреть отфильтрованные товары. В приложении представлены такие виды мебели, как шкафы, стулья, столы, комоды, тумбы, диваны, кровати и другие категории. Клиент может просмотреть каталог мебели, выбрать интересующую мебель и получить подробную информацию о нем. После выбора товара, он может заказать его. Также они могу заказать мебель с редактируемыми параметрами из представленных вариантов с помощью *3D*-модели. Заказ – это директивный документ, представляющий собой требование к поставщику поставить требуемые товары в необходимом количестве, необходимого качества в согласованные сроки. Клиенты могут просматривать информацию о своих заказах.

Для обеспечения качественного сервиса, *web*-приложение должно иметь возможность получать отзывы от клиентов, что позволит оценить уровень оказываемых услуг, поможет другим пользователям сделать правильный выбор и внести необходимые изменения в работу.

Одним из важных элементов предметной области является специалист по приему заказов, он отвечает за управление заказами (следит за их статусом). Также ему предоставлена возможность получать статистику по заказам на определённый промежуток времени, которую он может скачать.

Немаловажную роль играет менеджер по управлению запасами, который отвечает за управление товарами на складах.

Таким образом, разработка *web*-приложения по учету и мониторингу выполнения заказов фирмы по производству мебели на *ASP.NET Core MVC* должна обеспечить удобный и быстрый доступ к базе данных мебели, предоставить возможность заказать мебель, отследить состояние заказа, управлять складом, оставить отзывов и другие функции, необходимые для работы фирмы.

1. **Обзор и анализ существующих аналогов**

В современном мире производства мебели всё более важным становится эффективный учёт и мониторинг выполнения заказов. Для облегчения этого процесса и повышения качества обслуживания на рынке существуют аналогичные *web*-приложения для учёта и мониторинга выполнения заказов в области производства мебели, которые предоставляют похожие возможности для учета и регистрация заказов мебели, мониторинга заказов. Можно выделить несколько *web*-сайтов, которые не являются *web*-приложениями для учёта и мониторинга выполнения заказов, но имеют похожий функционал.

Согласно [2], сайт «Пинскдрев» предлагает каталог мебели с широким выбором разнообразных коллекций и модулей, которые можно посмотреть, не выходя из дома. В разделе «каталог мебели» представлены сотни карточек товаров, в которых подробно описаны все технические характеристики мебели, размеры, видео-обзоры, портфолио фото, а также новые *3D*-модели. Для более консервативных покупателей, которые привыкли расспрашивать продавца-консультанта о свойствах приобретаемой мебели, сайт предлагает бесплатную онлайн-консультацию специалиста, либо заказать бесплатный звонок менеджера. Интерфейс сайта «Пинскдрев» представлен на рисунке 1.1.

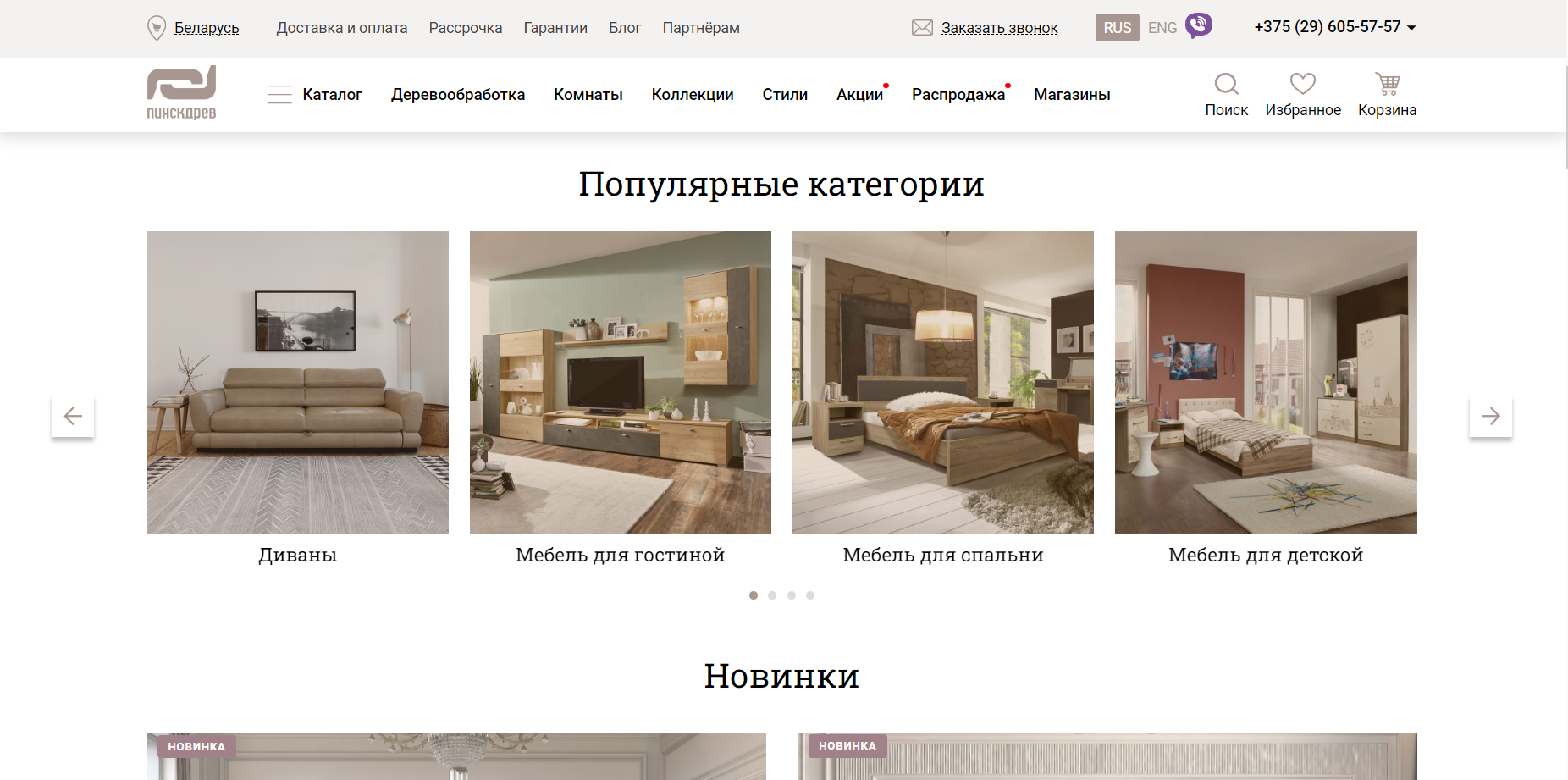


Рисунок 1.1 – Сайт для заказов мебели «Пинскдрев»

«*Divan.by*» – это онлайн-магазин, специализирующийся на продаже диванов в Минске. Они предлагают широкий выбор диванов различных стилей и размеров по доступным ценам. В каталоге магазина можно найти диваны с фотографиями и ценами. Согласно [3], они также предлагают скидки на определенные модели диванов. Интерфейс сайта «*Divan.by*» представлен на рисунке 1.2.

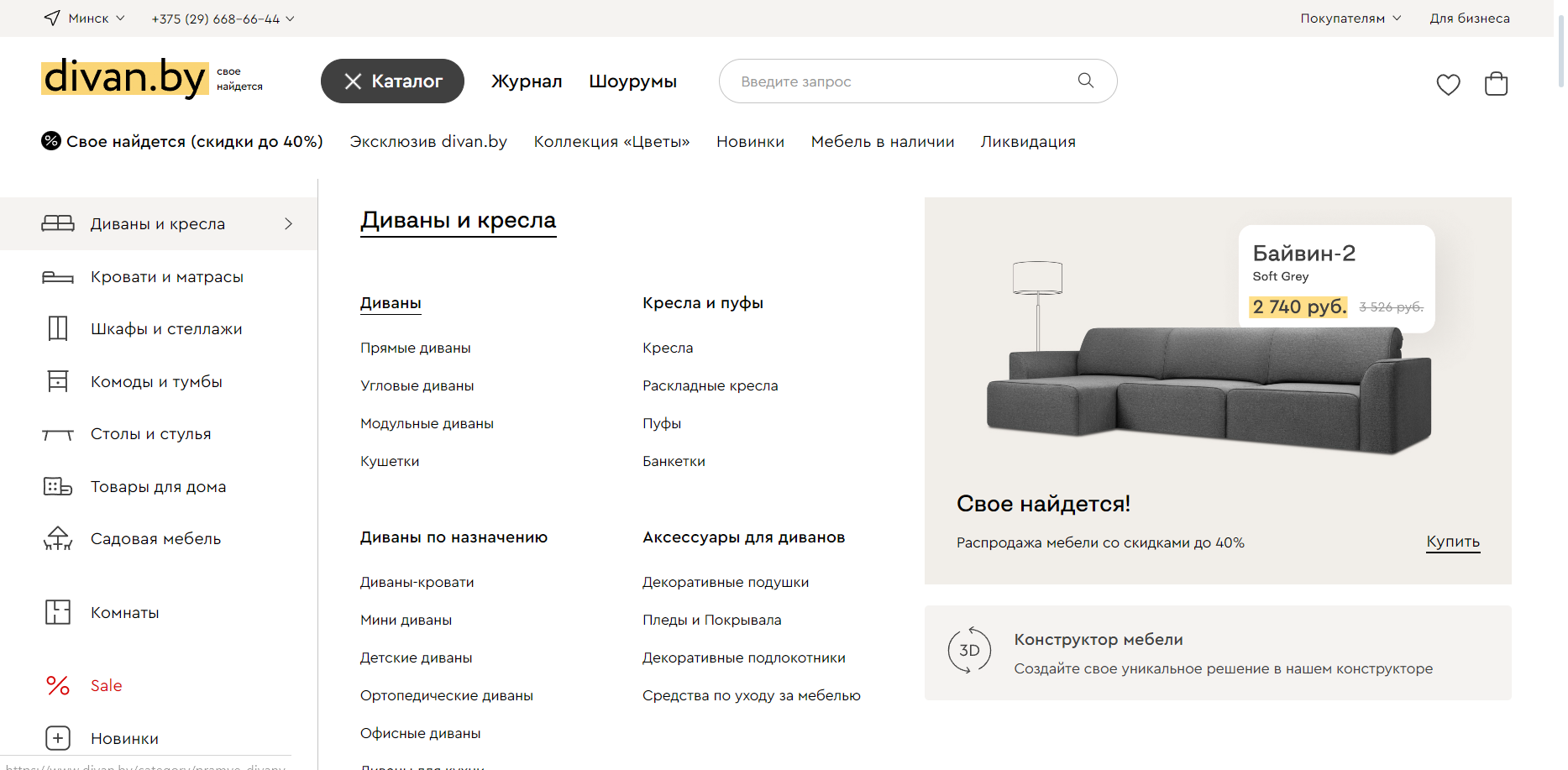


Рисунок 1.2 – Сайт для заказов мебели «*Divan.by*»

«*AKS*» – это интернет-магазин мебельной фурнитуры и комплектующих в Минске, где можно купить товары оптом и в розницу. На этом сайте [4] представлен широкий ассортимент функциональной и лицевой фурнитуры, крепежной фурнитуры, кромочных материалов, освещения и других сопутствующих товаров для мебельного производства и ремонта. Сайт также предлагает различные сервисы и поддержку, включая образовательную платформу, библиотеку *3D* моделей, калькуляторы расчетов и другие онлайн-сервисы. Контактная информация для оптовых и розничных продаж указана на сайте, а также есть возможность связаться с компанией по электронной почте или заказать обратный звонок. Интерфейс сайта «*AKS*» представлен на рисунке 1.3.

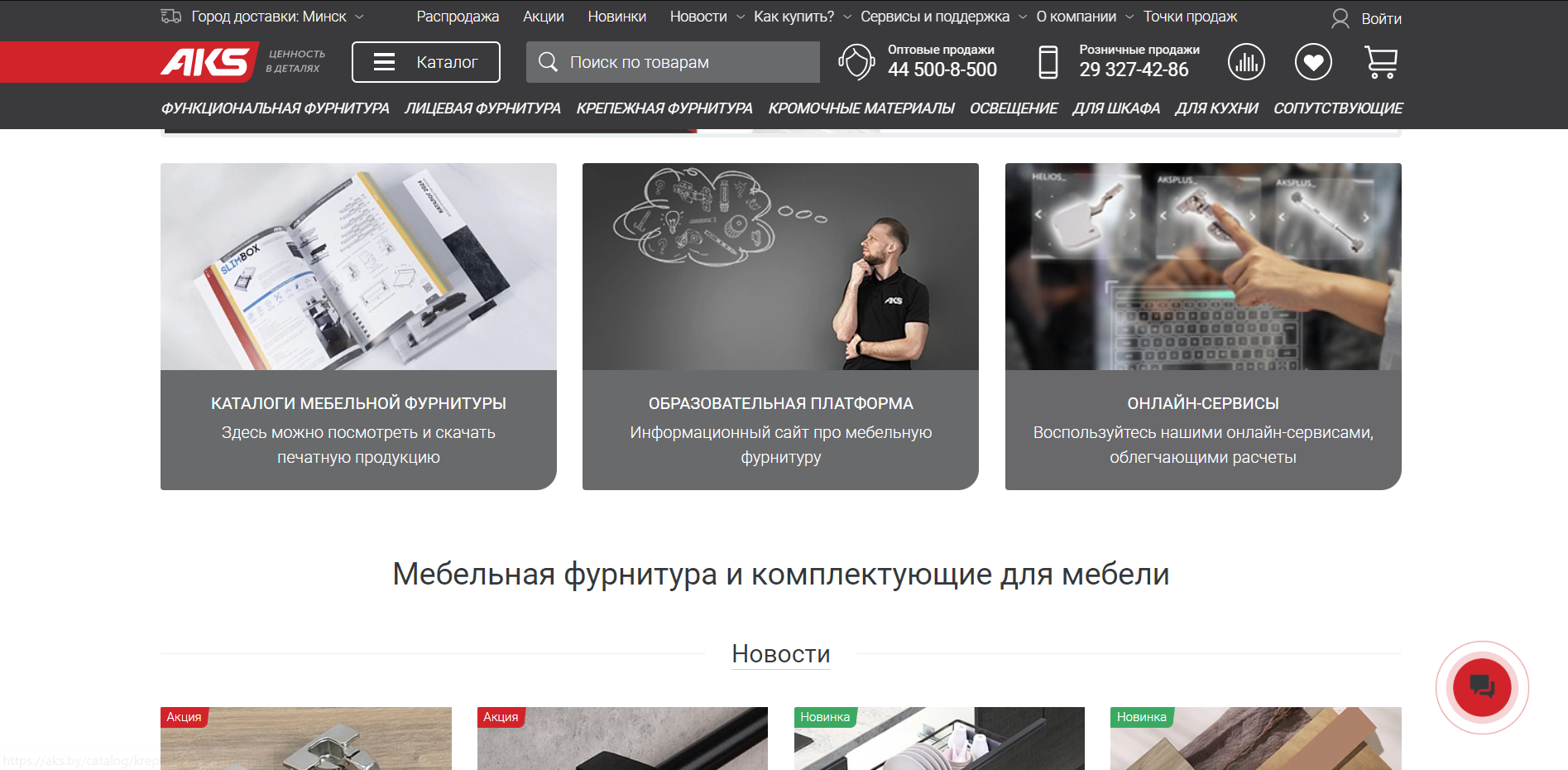


Рисунок 1.3 – Сайт для заказов мебели «*AKS*»

Согласно [5], сайт «Аксамит» предлагает широкий ассортимент мебели, включая гостиные, спальни, кухни, детские комнаты, матрасы, столы, стулья, шкафы и другие предметы мебели. Кроме того, на этом сайте предоставляются услуги доставки по Беларуси, возможность покупки в рассрочку и гарантийное обслуживание. Посетители могут связаться с магазином через указанные контактные данные или воспользоваться онлайн-консультацией. Интерфейс сайта «Аксамит» представлен на рисунке 1.4.



Рисунок 1.4 – Сайт для заказов мебели «Аксамит»

На сегодняшний день на рынке существует большое количество интернет-магазинов для заказа мебели, однако данное приложение позволит не только заказывать мебель, но и осуществлять учёт и мониторинг ваших заказов.

1. **Обзор технологий для реализации ПО**

Согласно [6], *web*-страницы описываются на языке гипертекстовой разметки – *HTML* (*HyperText Markup Language*). Он используется для описания структуры и содержимого *web*-документа, определяя, как браузер должен отображать элементы на странице. *HTML* состоит из различных элементов, которые определяют различные части *web*-страницы, такие как заголовки, параграфы, изображения, ссылки и многое другое. Основными компонентами *HTML* являются теги. Теги *HTML* являются ключевыми элементами, которые определяют тип и структуру содержимого *web*-страницы. Каждый тег обычно имеет открывающий и закрывающий тег, и содержимое между ними определяет, как будет отображаться элемент на *web*-странице. С помощью стандарта *CSS* (*Cascading Style Sheets*) создается отдельный документ, который называется таблицей стилей и определяет, как отдельные элементы в *HTML*-документе форматируются браузером.

Также используется *JavaScript* – облегченный язык программирования, разработанный для наполнения *web*-страниц интерактивными средствами. *JavaScript* является интерпретируемым языком, что означает, что код выполняется непосредственно в браузере без необходимости компиляции. Он предоставляет возможность добавлять динамическое поведение к *web*-страницам, обрабатывать события, взаимодействовать с пользователем и многое другое. *JavaScript* отлично подходит для создания всплывающих окон, встроенных анимационных эффектов или модификации контента, выводимого на *web*-страницу.

Для визуализации *3D*-моделей используется библиотека *JavaScript* [*Three.js*](https://en.wikipedia.org/wiki/Three.js), содержащая набор готовых классов для создания и отображения интерактивной *3D-*графики в *WebGL*. *WebGL* (*Web Graphics Library*) – это технология, которая позволяет использовать аппаратное обеспечение графического ускорения компьютера для отображения *3D*-графики в *web*-браузерах без необходимости установки плагинов или дополнительного программного обеспечения. *Three.js* облегчает разработку *3D*-графики на *web*-страницах, предоставляя высокоуровневый интерфейс и множество полезных функций.

Также для быстрого создания адаптивного дизайна сайта используется *Bootstrap* – бесплатный *HTML*/*CSS*/*JavaScript* фреймворк. *Bootstrap* позволяет с минимальными знаниями создавать интерфейсы, которые работают во всех популярных браузерах и не ломаются при изменении разрешения экрана. Благодаря тому, что в файлах *Bootstrap* прописаны самые распространенные стили и поведения для элементов, процесс верстки ускоряется в разы.

Средой разработки является *Visual Studio* –мощное средство разработчика, которое можно использовать для выполнения всего цикла разработки в одном месте. Это комплексная интегрированная среда разработки (*IDE*), которую можно использовать для записи, редактирования, отладки и сборки кода, а затем развертывания приложения. Помимо редактирования и отладки кода *Visual Studio* включает компиляторы, средства завершения кода, управление версиями,

расширения и многое другое, чтобы улучшить каждый этап процесса разработки

программного обеспечения.

Для разработки приложения использован язык программирования *C#*. Согласно [7], *C#* –это универсальный объектно-ориентированный язык программирования, разработанный к началу 2000-х годов. Он является одним из языков программирования, используемых для разработки приложений на платформе *.NET*. *C#* поддерживает основные принципы объектно-ориентированного программирования, такие как инкапсуляция, наследование и полиморфизм. Он позволяет создавать классы, объекты, абстрактные типы данных и использовать наследование для повторного использования кода. Также *C#* является статически типизированным языком, что означает, что каждая переменная и выражение имеют определенный тип, который проверяется на этапе компиляции. Это помогает предотвратить ошибки типов и улучшает надежность и безопасность программ. Язык *C#* поддерживает многопоточность, что позволяет создавать и управлять параллельными процессами в программе. Это полезно для создания многопоточных и асинхронных приложений, которые могут выполнять несколько задач одновременно.

*ASP.NET Core* является кроссплатформенной средой с открытым исходным кодом для создания современных приложений, оптимизированных для работы в облаке. Приложения *ASP.NET Core* отличаются простотой и модульностью, а также реализуют встроенную поддержку внедрения зависимостей, что позволяет расширить возможности тестирования и сопровождения. В сочетании с моделью *MVC*, которая поддерживает создание современных *web-API* в дополнение к приложениям на основе представлений, *ASP.NET Core* представляет собой эффективную платформу для разработки *web*-приложений корпоративного уровня.

Согласно [8], фреймворк *ASP.NET Core MVC* является частью платформы *ASP.NET Core*, его отличительная особенность – применение паттерна *MVC*. Преимуществом использования фреймворка *ASP.NET Core MVC* по сравнению с «чистым» *ASP.NET Core* является то, что он упрощает в ряде ситуаций и сценариев организацию и создание приложений, особенно это относится к большим приложениям. Взаимодействие пользователя с приложением *MVC* естественным образом следует циклу: пользователь предпринимает действие, в ответ на которое приложение изменяет свою модель данных и доставляет изменённое представление пользователю. Затем цикл повторяется. Это удобно укладывается в схему *web*-приложений, состоящих из последовательностей запросов и ответов *HTTP*.

Концепция паттерна *MVC* предполагает разделение логики приложения на три компонента: модель (*model*), представление (*view*), контроллер (*controller*).

Модель (*model*) описывает используемые в приложении данные, а также логику, которая связана непосредственно с данными, например, логику валидации данных. В ней находятся классы, которые отвечают за данные (сущности), манипулируют ними, записывают в базу данных и читают оттуда, а также за взаимодействие между самими сущностями. Если приложение небольшое, то, скорее всего, прямо здесь и будет происходить работа с данными.

Представление (*view*) отвечают за визуальную часть или пользовательский

интерфейс, нередко *html*-страница, через который пользователь взаимодействует

с приложением. Также представление может содержать логику, связанную с отображением данных. В то же время представление не должно содержать логику обработки запроса пользователя или управления данными.

Контроллер (*controller*) представляет центральный компонент *MVC*, который обеспечивает связь между пользователем и приложением, представлением и хранилищем данных. Он содержит логику обработки запроса пользователя. Контроллер получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления, наполненного данными моделей.

Для того, чтобы обновлять только нужные части страницы, без необходимости перезагрузки всей страницы, используется *AJAX* (*Asynchronous JavaScript and XML*). *AJAX* – это технология, которая позволяет асинхронно обмениваться данными между клиентской (браузерной) и серверной частями *web*-приложения. С помощью *AJAX* можно отправлять *HTTP* (*Hypertext Transfer Protocol*) -запросы с помощью *JavaScript* и получать обновленные данные. *AJAX* запросы позволяют передавать только необходимые данные, что может сократить объем передаваемой информации и улучшить производительность приложения.

Для работы с форматом *JSON* используется библиотека *Newtonsoft.Json*. *Newtonsoft.Json* обладает широкими возможностями настройки и гибкости, позволяя управлять процессом преобразования объектов в *JSON* и процессом преобразования *JSON* обратно в объекты, включая обработку сложных типов данных, настройку формата вывода и обработку ошибок. *Newtonsoft.Json* является одной из наиболее популярных и надежных библиотек для работы с *JSON* в *.NET*.

В качестве базы данных был выбран *MS SQL Server*. Согласно [9], *MS SQL Server* – это система управления реляционными базами данных, разработанная компанией *Microsoft*, работающая по клиент-серверной модели. Она используется для хранения и управления структурированными и неструктурированными данными в различных приложениях, включая корпоративные системы, *web*-приложения, приложения для анализа данных и многое другое. Реляционные базы – это такие базы, где все данные хранятся в виде таблиц, и эти таблицы связаны между собой: одна с другой, другая – с третьей и так далее. Реляционный подход хорош тем, что позволяет СУБД избежать массы лишней работы.

Преимущества использования *MS SQL Server*:

* *MS SQL Server* хорошо интегрируется *Visual Studio*;
* *MS SQL Server* предлагает богатый набор встроенных функций и возможностей, включая поддержку транзакций, хранимые процедуры, триггеры, аналитические инструменты и многое другое;
* система обладает встроенными средствами обеспечения целостности данных и защиты от сбоев;
* система имеет возможность горизонтального и вертикального масштабирования, что позволяет адаптировать ее под потребности бизнеса.

Для доступа к данным используется технология *Entity Framework Core*, которая, согласно [10], позволяет разработчикам создавать запросы к базе данных на языке *LINQ (Language Integrated Query)*, что упрощает написание запросов и улучшает читаемость кода. Она также обладает встроенными возможностями по кэшированию данных, что позволяет ускорить работу приложения и уменьшить нагрузку на базу данных. Её использование в проекте позволяет сократить время на разработку и упростить поддержку приложения, обеспечивая быстрый доступ к данным и более эффективное использование ресурсов базы данных. *Entity Framework Core* позволяет разработчикам создавать объектные модели данных, называемые моделями сущностей (*Entity Models*), и автоматически отображать их на структуру базы данных. Также он позволяет легко изменять структуру базы данных с помощью механизма миграций. Разработчики могут создавать миграции, которые представляют изменения в модели данных, и автоматически применять эти изменения к базе данных. Это упрощает процесс развития приложения и обновления базы данных. *Entity Framework Core* предоставляет специальный контекст данных для тестирования, который позволяет разработчикам создавать и заполнять тестовую базу данных в памяти, что упрощает написание и выполнение автономных модульных тестов для приложений, использующих *Entity Framework Core*.

Для ускорения и упрощения сопоставления объектов используется библиотека *Automapper*. Она позволяет автоматически копировать данные из одного объекта в другой, основываясь на сопоставлении их свойств. *Automapper* освобождает разработчика от рутинной работы по копированию значений свойств вручную. Также эта библиотека позволяет задавать пользовательские правила сопоставления, если стандартные правила не подходят. Это включает в себя преобразование типов данных, игнорирование определенных свойств или настройку обратного сопоставления.

Для создания системы авторизации и аутентификации используется встроенный во фреймворк *ASP.NET Core* модуль *Identity*. *ASP.NET Core Identity* представляет встроенную в *ASP.NET* систему аутентификации и авторизации. Данная система позволяет пользователям создавать учетные записи, аутентифицироваться, управлять учетными записями или использовать для входа на сайт учетные записи внешних провайдеров, таких как *Facebook*, *Google*, *Microsoft*, *Twitter* и других. Модуль *Identity* предоставляет интерфейсы и классы для управления пользователями и их ролями в приложении. Разработчики могут создавать, удалять, обновлять и искать пользователей, а также назначать и удалять роли для каждого пользователя. *ASP.NET Core Identity* может хранить данные пользователей и их ролей в различных источниках данных, таких как базы данных *SQL Server*, *SQLite*, *MySQL* или *PostgreSQL*. Он также поддерживает настраиваемые провайдеры хранения данных, что позволяет разработчикам интегрировать его с существующими системами хранения данных.

1. **Постановка задачи**

В результате проведённого анализа предметной области, обзора и анализа существующих аналогов, обзора технологий для реализации ПО должен быть сформулирован список требований, которые нужно выполнить при разработке *web*-приложения. Выполнение данных требований в конечном и готовом к использованию программном продукте будет свидетельствовать о том, что возложенные на дипломную работу цели были успешно достигнуты, а поставленные задачи решены.

Необходимо разработать *web*-приложение на *C# ASP.NET Core MVC* для учета и мониторинга выполнения заказов фирмы по производству мебели, которое позволит клиентам осуществлять поиск мебели, задавать параметры и заказывать мебель, оставлять отзывы, поможет сотрудникам следить за заказами и вести их учёт. Для реализации данной задачи нужно реализовать сервер приложения и его клиентскую часть.

Для достижения этой цели необходимо решить следующие задачи:

* разработка системы авторизации и регистрации пользователей;
* реализация учета и регистрации заказов мебели включая информацию о клиенте, выбранной мебели, деталях заказа;
* реализация возможности отслеживания статуса выполнения каждого заказа;
* реализация фильтрации мебели по категориям;
* реализация возможности поиска мебели по заданному названию для удобства клиентов при выборе продукта;
* реализация возможности получения уведомлений для клиентов об изменениях в статусе и ходе выполнения заказа;
* реализация возможности для клиентов оставлять отзывы о мебели, качестве обслуживания и общем опыте работы с фирмой.
* расчёт статистики по заказам, включая информацию о количестве выполненных заказов, популярности определенных товаров и других показателях.
* реализация возможности учёта и управления запасами склада для контроля наличия товаров.

В результате выполнения данной работы будет разработано *web*-приложение на *C# ASP.NET Core MVC* для учёта и мониторинга выполнения заказов фирмы по производству мебели. Это приложение будет предоставлять компании возможность эффективно отслеживать этапы выполнения заказа, начиная от его размещения и заканчивая доставкой клиенту. Приложение будет обладать удобным интерфейсом, интуитивно понятными функциями и возможностью доступа через *web*-браузер.

1. **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**
2. **Функциональное моделирование**

Функциональное моделирование – это методология разработки ПО, которая основывается на анализе функций, выполняемых системой, и определении, как эти функции будут реализованы. Оно заключается в определении функций, которые должна выполнять система, и разбиении их на более мелкие подзадачи для более удобной и эффективной реализации. Согласно [11], для формирования представления о предметной области используют *UML*-диаграммы, которые представлены на рисунках.

Актер «Гость» представляет незарегистрированного пользователя, который имеет ограниченные возможности в системе. Его основные действия включают регистрацию, авторизацию, просмотр информации о мебели, просмотр отзывов и поиск товаров. Прецеденты актера «Гость» представлены на рисунке 2.1.

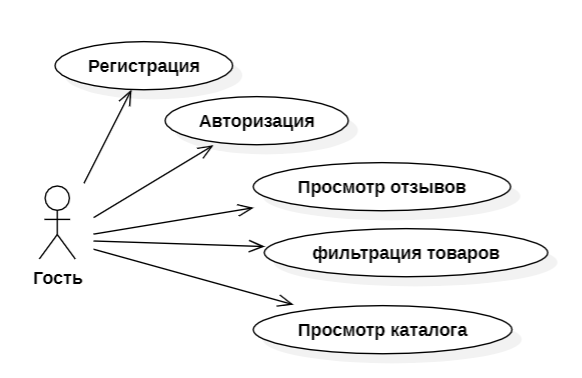


Рисунок 2.1 – Прецеденты актера «Гость»

Актер «Клиент» представляет зарегистрированного пользователя *web*-приложения. Он может выполнять те же действия, что и актер «Гость», а также выходить из аккаунта, заказывать товары, следить за своими заказами, оставлять отзыв, получать уведомления.

Прецеденты актера «Клиент» представлены на рисунке 2.2.

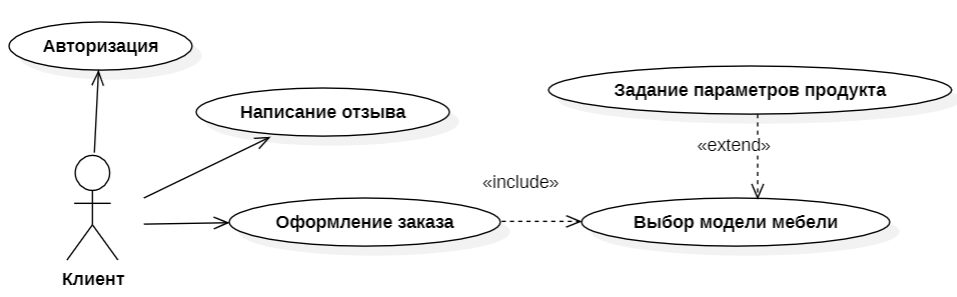


Рисунок 2.2 – Прецеденты актера «Клиент»

Актер «Специалист по приему заказов» может просматривать весь список заказов, обновлять статус заказов, а также получать статистику за промежуток времени.

Прецеденты актера «Специалист по приему заказов» представлены на рисунке 2.3.



Рисунок 2.3 – Прецеденты актера «Специалист по приему заказов»

Актер «Менеджер по управлению запасами» может добавлять новый товар на склад, добавлять существующий товар на склад, удалять товар со склада, а также редактировать товар.

Прецеденты актера «Менеджер по управлению запасами» представлены на рисунке 2.4.

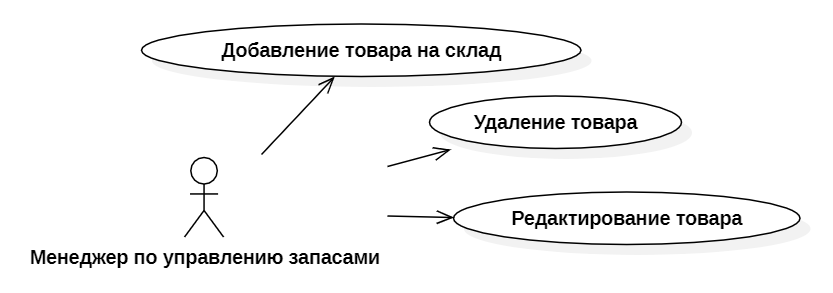


Рисунок 2.4 – Прецеденты актера «Менеджер по управлению запасами»

Актер «Администратор» может добавлять, удалять, просматривать список ролей и назначать роли пользователям.

Прецеденты актера «Администратор» представлены на рисунке 2.5.



Рисунок 2.5 – Прецеденты актера «Администратор»

1. **Информационное обеспечение приложения**

Информационное обеспечение приложения включает в себя базу данных, которая, согласно [12], обеспечивает хранение и доступ к информации (в данной работе о мебели, пользователях, и других важных данных, необходимых для работы приложения).

Описание таблиц базы данных представлено в таблице 2.1.

Таблица 2.1 – Описание таблиц базы данных

|  |  |
| --- | --- |
| Название таблицы | Назначение |
| *Products* | Хранение информации о мебели |
| *ParameterProducts* | Хранение связи между товарами и параметрами |
| *Parameters* | Хранение информации о параметрах |
| *ProductGroups* | Хранение информации о товарных группах |
| *Warehouses* | Хранение информации о характеристике склада |
| *WarehouseProducts* | Хранение информации о складах |
| *Orders* | Хранение информации о заказах |
| *Clients* | Хранение информации о клиентах |
| *Reviews* | Хранение информации об отзывах |
| *Providers* | Хранение информации о поставщиках деталей |
| *Notifications* | Хранение информации об уведомлениях |

Описание атрибутов таблицы «*Products*» представлено в таблице 2.2.

Таблица 2.2 – Описание атрибутов таблицы «*Products*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |
| *Name* | *Nvar-char(MAX)* | Хранение названия мебели | Нет | Нет |
| *Description* | *Nvar-char(MAX)* | Хранение описания мебели | Нет | Нет |
| *Type* | *Nvar-char(MAX)* | Хранение типа мебели | Нет | Нет |
| *Cost* | *Float* | Хранение стоимости мебели | Нет | Нет |
| *ProviderId* | *Int* | Хранение идентификатора поставщика | Нет | Да |
| *ProviderId* | *Int* | Хранение идентификатора поставщика | Нет | Да |
| *Product­GroupId* | *Int* | Хранение идентификатора товарной группы | Нет | Да |

Таблица «*ParameterProducts*» в базе данных является справочником-связкой между параметрами и товарами. Описание атрибутов таблицы «*ParameterProducts*» представлено в таблице 2.3.

Таблица 2.3 – Описание атрибутов таблицы «*ParameterProducts*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| 1 | 2 | 3 | 4 | 5 |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |

Продолжение таблицы 2.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| *ProductId* | *Int* | Хранение идентификатора товара | Нет | Да |
| *ParameterId* | *Int* | Хранение идентификатора параметра | Нет | Да |

Таблица «*Parameters*» в базе данных является справочником. В ней представлены поля названий параметров и их значений. Описание атрибутов таблицы «*Parameters*» представлено в таблице 2.4.

Таблица 2.4 – Описание атрибутов таблицы «*Parameters*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |
| *Name* | *Nvar-char(MAX)* | Хранение названия параметра | Нет | Нет |
| *Value* | *Float* | Хранение значения параметра | Нет | Нет |

Таблица «*ProductGroups*» в базе данных является справочником. В ней представлены названия товарной группы. Описание атрибутов таблицы «*ProductGroups*» представлено в таблице 2.5.

Таблица 2.5 – Описание атрибутов таблицы «*ProductGroups*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |
| *Name* | *Nvar-char(MAX)* | Хранение названия группы | Нет | Нет |

Таблица «*Warehouses*» в базе данных является справочником. Она содержит номер склада, материал, цвет, название картинки. Описание атрибутов таблицы «*Warehouses*» представлено в таблице 2.6.

Таблица 2.6 – Описание атрибутов таблицы «*Warehouses*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |
| *WarehouseNumber* | *Int* | Хранение номера склада | Нет | Нет |
| *Material* | *Nvar-char(MAX)* | Хранение названия материала | Нет | Нет |
| *Color* | *Nvar-char(MAX)* | Хранение названия цвета | Нет | Нет |
| *ImageURL* | *Nvar-char(MAX)* | Хранение названия картинки | Нет | Нет |

Таблица «*WarehouseProducts*» в базе данных является справочником-связкой между складом и товаром, также она содержит количество товара.

Описание атрибутов таблицы «*WarehouseProducts*» представлено в таблице 2.7.

Таблица 2.7 – Описание атрибутов таблицы «*WarehouseProducts*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |
| *Count* | *Int* | Хранение количества товара | Нет | Нет |
| *ProductId* | *Int* | Хранение идентификатора товара | Нет | Да |
| *WarehouseId* | *Int* | Хранение идентификатора склада | Нет | Да |

Таблица «*Orders*» в базе данных является справочником-связкой. Она содержит данные о заказах, прогресс заказа, и также содержит идентификаторы клиента и товара. Эти идентификаторы используются для связи с таблицами клиента и товара.

Описание атрибутов таблицы «*Orders*» представлено в таблице 2.8.

Таблица 2.8 – Описание атрибутов таблицы «*Orders*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |
| *Date* | *Datetime2(7)* | Хранение даты заказа | Нет | Нет |
| *ProgressCount* | *Int* | Хранение значения прогресса заказа | Нет | Нет |
| *ProductId* | *Int* | Хранение идентификатора товара | Нет | Да |
| *TotalCost* | Float | Хранение стоимости | Нет | Нет |
| *Status* | *Nvar-char(MAX)* | Хранение статуса заказа | Нет | Нет |
| *ClientId* | *Int* | Хранение идентификатора клиента | Нет | Да |

Таблица «*Clients*» в базе данных является справочником. Описание атрибутов таблицы «*Clients*» представлено в таблице 2.9.

Таблица 2.9 – Описание атрибутов таблицы «*Clients*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |
| *Picture* | *Nvar-char(MAX)* | Хранение названия картинки клиента | Нет | Нет |
| *Surname* | *Nvar-char(MAX)* | Хранение ФИО клиента | Нет | Нет |
| *Phone* | *Nvar-char(MAX)* | Хранение номера телефона | Нет | Нет |
| *Address* | *Nvar-char(MAX)* | Хранение адреса доставки | Нет | Нет |

Описание атрибутов таблицы «*Reviews*» представлено в таблице 2.10.

Таблица 2.10 – Описание атрибутов таблицы «*Reviews*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |
| *Comment* | *Nvar-char(MAX)* | Хранение текста комментария | Нет | Нет |
| *Rating* | *Int* | Хранение оценки | Нет | Нет |
| *Date* | *Datetime2(7)* | Хранение даты отзыва | Нет | Нет |
| *ClientId* | *Int* | Хранение идентификатора клиента | Нет | Да |

Таблица «*Providers*» в базе данных является справочником.

Описание атрибутов таблицы «*Providers*» представлено в таблице 2.11.

Таблица 2.11 – Описание атрибутов таблицы «*Providers*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |
| *Name* | *Nvar-char(MAX)* | Хранение названия поставщика деталей | Нет | Нет |
| *Surname* | *Nvar-char(MAX)* | Хранение ФИО представителя поставщика деталей | Нет | Нет |
| *Address* | *Nvar-char(MAX)* | Хранение адреса | Нет | Нет |
| *Phone* | *Nvar-char(MAX)* | Хранение номера телефона | Нет | Нет |

Таблица «*Notifications*» в базе данных является справочником-связкой. Она содержит внешний ключ клиента. Описание атрибутов таблицы «*Notifications*» представлено в таблице 2.12.

Таблица 2.12 – Описание атрибутов таблицы «*Notifications*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение | Первичный ключ | Внешний ключ |
| *Id* | *Int* | Хранение номера записи в таблице | Да | Нет |
| *Date* | *Datetime2(7)* | Хранение даты уведомления | Нет | Нет |
| *Message* | *Nvar-char(MAX)* | Хранение текста уведомления | Нет | Нет |
| *Status* | *Nvar-char(MAX)* | Хранение статуса уведомления | Нет | Нет |
| *ClientId* | *Int* | Хранение идентификатора клиента | Нет | Да |

На рисунке 2.6 приведена схема базы данных, которая была реализована с применением реляционной базы данных *SqlServer*.

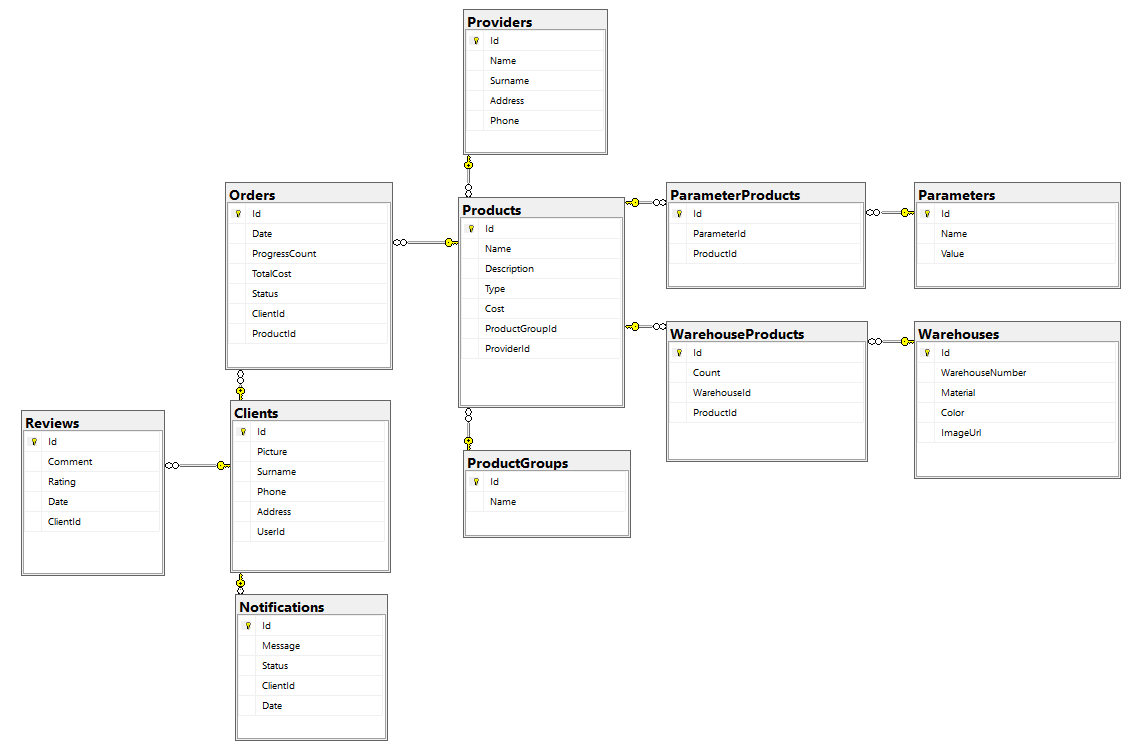


Рисунок 2.6 – Структура базы данных

Описание связей между сущностями:

* связь между таблицами «*Clients*» и «*Reviews*» реализуется через внешний ключ «*ClientId*»;
* связь между таблицами «*Clients*» и «*Notifications*» реализуется через внешний ключ «*ClientId*»;
* связь между таблицами «Clients» и «*Orders*» реализуется через внешний ключ «*ClientId*»;
* связь между таблицами «*Products*» и «*Orders*» реализуется через внешний ключ «*ProductId*»;
* связь между таблицами «Products» и «*ProductGroups*» реализуется через внешний ключ «*ProductGroup*Id»;
* связь между таблицами «*ProductsWarehouses*» и «*Products*» реализуется через внешний ключ «*ProductId*»;
* связь между таблицами «*ProductsWarehouses*» и «*Warehouses*» реализуется через внешний ключ «*Warehouse*Id»;
* связь между таблицами «*ProductsParameters*» и «*Products*» реализуется через внешний ключ «*ProductId*»;
* связь между таблицами «*ProductsParameters*» и «*Parameters*» реализуется через внешний ключ «*ParameterId*»;
* связь между таблицами «*Providers*» и «*Products*» реализуется через внешний ключ «*ProviderId*».

Все связи точно не имеют каскадного ограничения целостности.

1. **Архитектура приложения**

*Web*-приложение имеет архитектуру клиент-сервер. Его код разделен на два компонента: скрипты на стороне клиента и скрипты на стороне сервера. Характеристика клиент-сервер описывает отношения взаимодействующих программ в приложении.

Скрипт на стороне клиента отвечает за функциональность пользовательского интерфейса, например, кнопки и выпадающие поля. Когда конечный пользователь нажимает на ссылку *web*-приложения, *web*-браузер загружает скрипт на стороне клиента и отображает графические элементы и текст для взаимодействия с пользователем. Например, пользователь может читать контент, смотреть видео или заполнять данные в форме контакта. Такие действия, как нажатие кнопки отправки, передаются на сервер в виде клиентского запроса.

Скрипт на стороне сервера обрабатывает данные. Сервер *web*-приложений обрабатывает запросы клиентов и отправляет ответ. Обычно запросы касаются получения дополнительных данных или изменения или сохранения новых данных. Например, если пользователь нажимает на кнопку «Подробнее», сервер *web*-приложений отправляет контент обратно пользователю. Если пользователь нажмет кнопку «Отправить», сервер приложений сохранит данные пользователя в базе данных. В некоторых случаях сервер завершает запрос данных и отправляет полную *HTML*-страницу обратно клиенту. Это называется рендерингом на

на стороне сервера.

Архитектура данного приложения представляет собой трехуровневую клиент-серверную архитектуру. В этой архитектуре всё приложение организовано на трех вычислительных уровнях:

* слой пользовательского интерфейса (*User Interface*);
* слой бизнес-логики (*Business Logic Layer*);
* слой доступа к данным (*Data Access Layer*).

*WebAppFurnitureOrders.UI* – это слой пользовательского интерфейса, который является самым верхним уровне в архитектуре. Его назначение – принимать запросы от клиента и отображать информацию клиенту. Здесь определены контроллеры, представления и модели представления, которые обрабатывают запросы пользователей, взаимодействуют с слоем *BLL* для получения данных и отображают информацию в пользовательском интерфейсе. Если говорить о *web*-уровнях, то они разрабатываются с использованием таких языков, как [*HTML*](https://translated.turbopages.org/proxy_u/en-ru.ru.25f1cfa3-65eefe98-590a5fb6-74722d776562/https/www.geeksforgeeks.org/html/), [*CSS*](https://translated.turbopages.org/proxy_u/en-ru.ru.25f1cfa3-65eefe98-590a5fb6-74722d776562/https/www.geeksforgeeks.org/css/), [*JavaScript*.](https://translated.turbopages.org/proxy_u/en-ru.ru.25f1cfa3-65eefe98-590a5fb6-74722d776562/https/www.geeksforgeeks.org/introduction-to-javascript/)

*WebAppFurnitureOrders.BLL* – это слой бизнес-логики, средний уровень архитектуры, здесь подробно обрабатывается информация либо запрос, собираемые через уровень представления. Он взаимодействует с сервером *WebAppFurnitureOrders.DAL*, на котором хранятся данные. Он обрабатывает запрос клиента, форматирует его и отправляет обратно клиенту. С использованием среднего уровня осуществляется связь между уровнем представления и уровнем данных, то есть *BLL* является мостом между этими слоями.

*WebAppFurnitureOrders.DAL* – это слой доступа к данным, являющийся последним уровнем архитектуры, также известный как уровень базы данных. Он используется для хранения обработанной информации, чтобы ее можно было получить позже, когда потребуется. Она состоит из сервера баз данных *MS SQL*. Здесь определены модели данных, контекст базы данных и репозитории для выполнения операций чтения и записи данных.

Трёхслойная архитектура является популярным и эффективным подходом к разработке приложений, который предполагает разделение кода на слои. Каждый из этих слоев выполняет свои чётко определённые обязанности, что способствует улучшенной масштабируемости, производительности и доступности приложения.

# **СТРУКТУРА И РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **Описание реализации программного комплекса**

Программный комплекс имеет трехслойную архитектуру, состоящую из компонентов *DAL*, *BLL* и *UI*. Каждый слой связан с предыдущим при помощи зависимостей. Если провести анализ архитектуры проекта, то можно построить зависимость проектов друг от друга. На рисунке 3.1 представлена архитектура проекта.

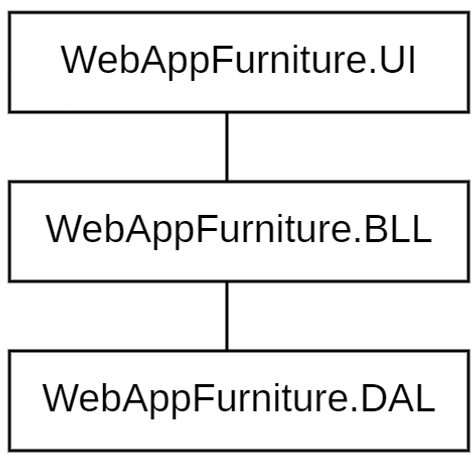


Рисунок 3.1 – Архитектура приложения «*WebAppFurniture*»

На рисунке 3.1 можно увидеть, что каждый модуль проекта связан друг с другом и может использовать его функционал.

*WebAppFurniture.DAL* включает в себя модели данных, контекст базы данных и репозитории. Модели данных определяют структуру данных, которые будут использоваться при работе с базой данных. Контекст базы данных устанавливает соединение с базой данных и выполняет операции чтения и записи данных. Репозитории предоставляют удобные методы для выполнения операций с данными, абстрагируя детали доступа к данным от остальных компонентов приложения. Для связи классов и таблиц из базы данных используется интерфейс *IRepository*, который реализуется классами, использующими модели для работы с информацией.

В таблице 3.1 представлены основные методы интерфейса *IRepository*.

Таблица 3.1 – Методы класса «IRepository»

|  |  |
| --- | --- |
| Название метода | Описание |
| 1 | 2 |
| *GetAllAsync* | Извлечь все данные из базы данных |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| 1 | 2 |
| *CreateAsync* | Добавить информацию в базу данных |
| *UpdateAsync* | Обновить информацию из базы данных по идентификатору |
| *DeleteAsync* | Удалить информацию из базы данных по идентификатору |
| *GetByIdAsync* | Извлечь информацию из базы данных по идентификатору |

Все классы, которые реализуют этот интерфейс используют эти методы для работы с базой данных. Также, данные классы используются в бизнес-слое. Реализацию этих методов можно увидеть в Приложении А.

*WebAppFurniture.BLL* обрабатывает запросы от *UI*, выполняет необходимые операции с данными, осуществляет валидацию и обработку ошибок, а также реализует логику бизнес-процессов, связанных с функциональностью *web* -приложения. В таблице 3.2 представлены основные классы, которые используются для обработки пользовательских запросов.

Таблица 3.2 – Классы слоя бизнес-логики

|  |  |
| --- | --- |
| Название класса | Описание |
| *ClientService* | Класс бизнес-слоя для работы с данными клиентов |
| *NotificationService* | Класс бизнес-слоя для работы с данными об уведомлениях |
| *OrderService* | Класс бизнес-слоя для работы с данными о заказах |
| *ParameterProductService* | Класс бизнес-слоя для работы с данными о |
| *ParameterService* | Класс бизнес-слоя для работы с данными о параметрах |
| *ProductGroupService* | Класс бизнес-слоя для работы с данными о товарной группе |
| *ProductService* | Класс бизнес-слоя для работы с данными о товарах |
| *ProviderService* | Класс бизнес-слоя для работы с данными о поставщиках деталей |
| *ReviewService* | Класс бизнес-слоя для работы с данными об отзывах |
| *WarehouseProductService* | Класс бизнес-слоя для работы с данными о складах |
| *WarehouseService* | Класс бизнес-слоя для работы с данными о характеристике склада |

Методы классов, представленные в таблице 3.2 используются в слое отображения для организации взаимодействия пользователя с приложением. Реализацию этих классов можно увидеть в Приложении Б.

*WebAppFurniture.UI* представляет собой *web*-интерфейс приложения, через который пользователи взаимодействуют с системой. Он реализован с использованием *ASP.NET Core MVC,* что обеспечивает удобное разделение логики отображения и обработки запросов. *UI* отображает информацию о различной мебели, предоставляет формы для регистрации и входа пользователей, позволяет выполнять поиск мебели, оформлять заказы, получать уведомления, управлять заказами и складом, а также оставлять отзывы.

В таблице 3.3 представлены основные контроллеры слоя отображения.

Таблица 3.3 – Основные контроллеры слоя отображения

|  |  |
| --- | --- |
| Название контроллера | Описание |
| *Account* | Данный контроллер занимается обработкой запросов к страницам, связанным с аккаунтами |
| *Home* | Данный контроллер занимается обработкой запросов к страницам, связанным с главной страницей |
| *Notification* | Данный контроллер занимается обработкой запросов к страницам, связанным с уведомлениями |
| *Order* | Данный контроллер занимается обработкой запросов к страницам, связанным с заказами |
| *Review* | Данный контроллер занимается обработкой запросов к страницам, связанным с отзывами |
| *Roles* | Данный контроллер занимается обработкой запросов к страницам, связанным с ролями аккаунтов |
| *Warehouse* | Данный контроллер занимается обработкой запросов к страницам, связанным со складом |

Контроллеры, представленные на таблице 3.3, играют важную роль в слое отображения. Они отвечают за обработку запросов пользователей, валидацию данных, взаимодействие с бизнес-логикой и предоставление соответствующих представлений.

Таким образом, проведя анализ программного комплекса приложения было описано как устроено приложения и то, какие модули и классы используются для организации работы приложения и взаимодействия с пользователем. Реализацию этих контроллеров можно увидеть в Приложении В.

## **Описание пользовательского интерфейса**

Описание пользовательского интерфейса включает описание визуальных элементов, компонентов и механизмов, которые пользователь может видеть и использовать при работе с приложением. Это может включать в себя размещение элементов на экране, цветовую схему, шрифты, кнопки, поля ввода, меню и другие элементы интерфейса. Кроме визуальных аспектов, описание пользовательского интерфейса также должно включать описание функциональности и возможностей, доступных пользователю.

Пользовательский интерфейс для незарегистрированного пользователя предоставляет удобные инструменты для ознакомления с *web*-приложением. На главной странице для незарегистрированного пользователя доступен просмотр доступной мебели, фильтрация мебели, поиск мебели, просмотр *3D*-модели с параметрами, просмотр отзывов и кнопка «Войти». На рисунке 3.2 представлена главная страница для незарегистрированного пользователя.

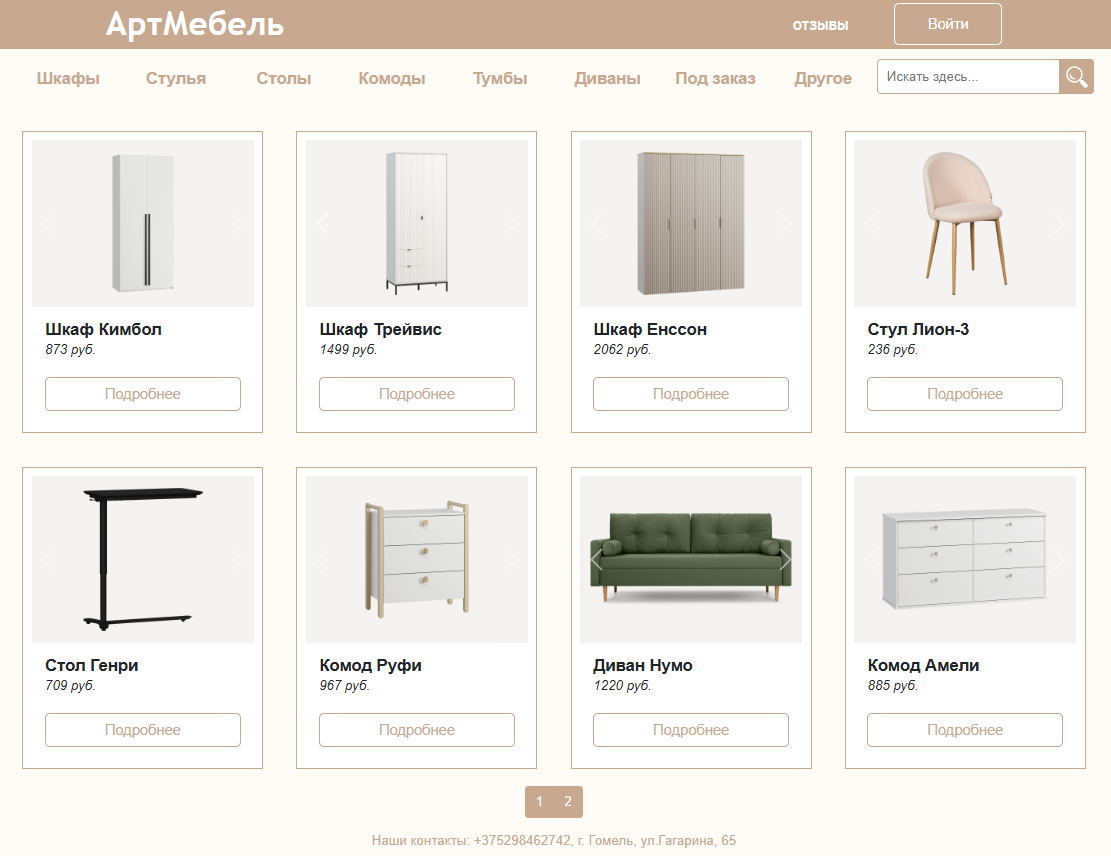


Рисунок 3.2 – Главная страница для незарегистрированного пользователя

При нажатии на определенную категорию мебели на главной странице, пользователю будет показан список отфильтрованной мебели. При нажатии надпись «АртМебель» пользователь вернется на главную страницу. На рисунке 3.3 представлена главная страница для незарегистрированного пользователя по-

сле фильтрации мебели.

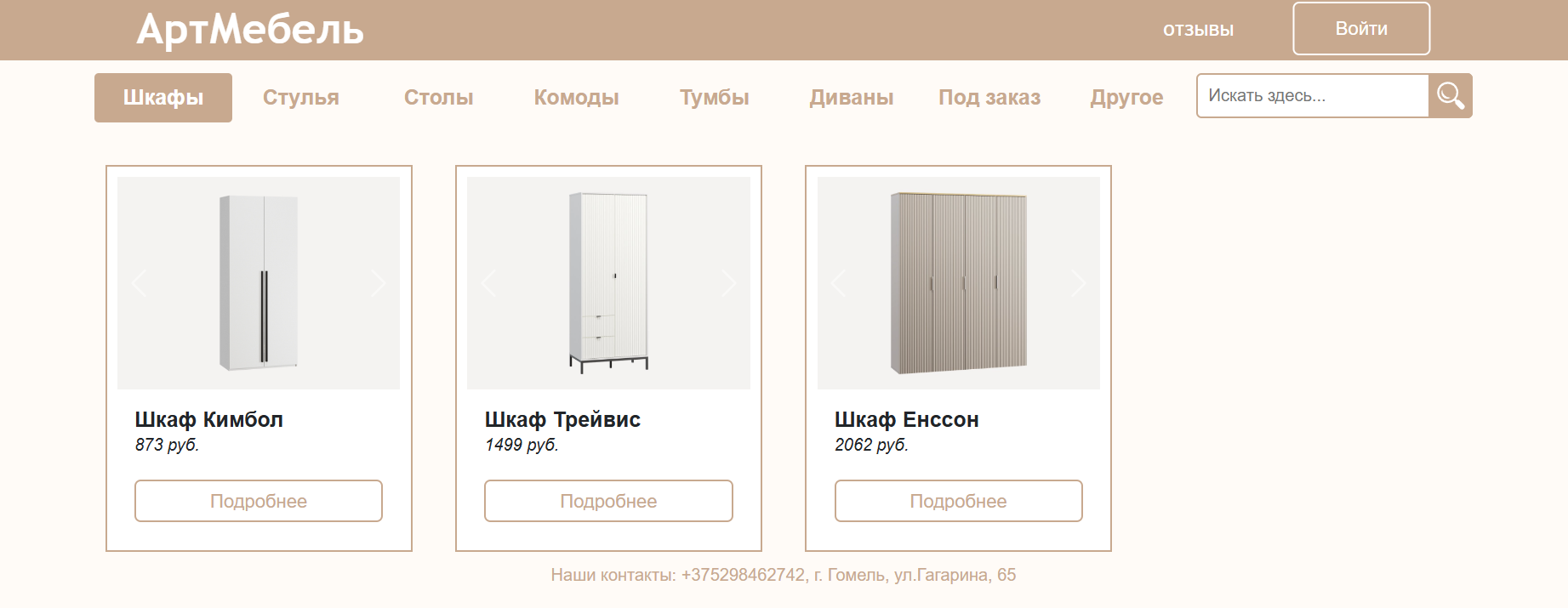


Рисунок 3.3 – Главная страница для незарегистрированного пользователя после фильтрации мебели

После ввода мебели в поисковую строку и нажатия на кнопку поиска пользователю выведется найденная мебель. На рисунке 3.4 представлена главная страница для незарегистрированного пользователя после поиска мебели.

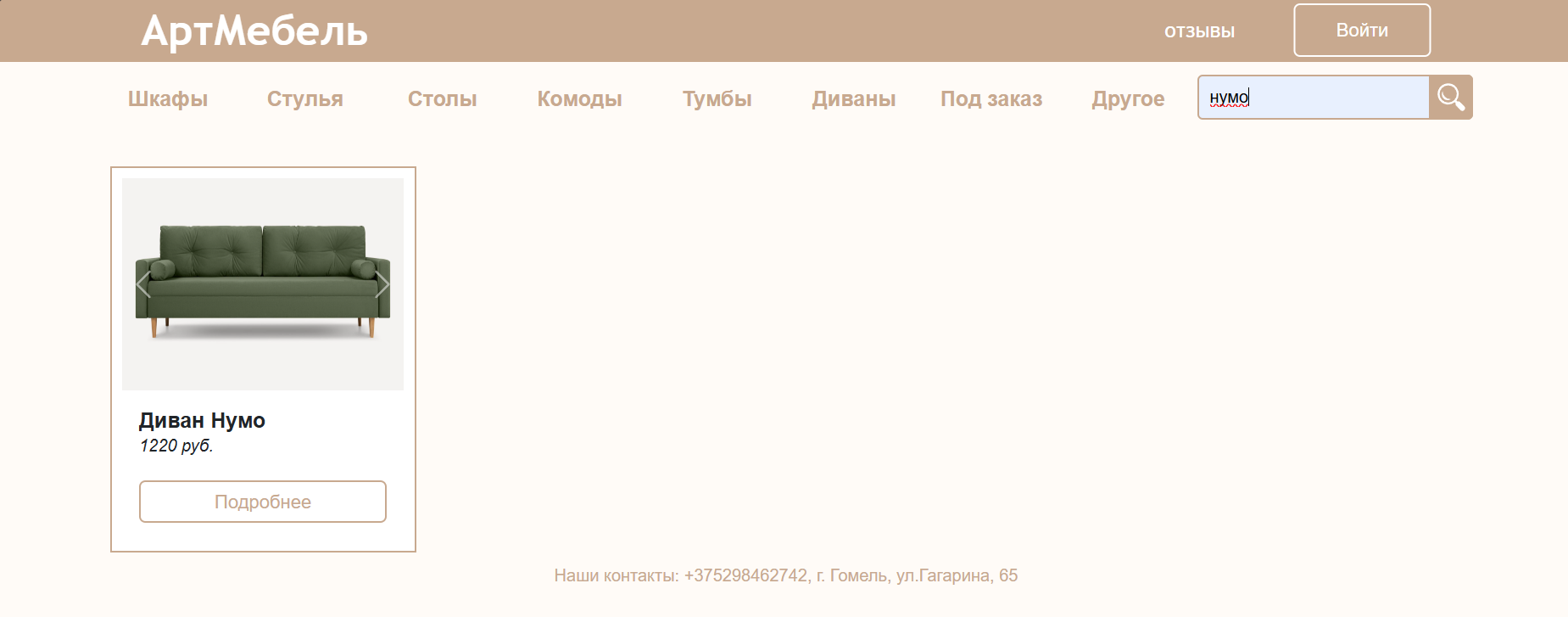


Рисунок 3.4 – Главная страница для незарегистрированного пользователя после поиска мебели

В каждой карточке мебели доступна кнопка «Подробнее». После нажатия на эту кнопку пользователь перейдет на страницу с подробностями товара. На ней доступен выбор цвета мебели, описание мебели, цена, а также кнопка «Войдите, чтобы сделать заказ», при нажатии на которую пользователя перебросит на страницу авторизации.

На рисунке 3.5 показана страница с подробностями товара для незарегистрированного пользователя.

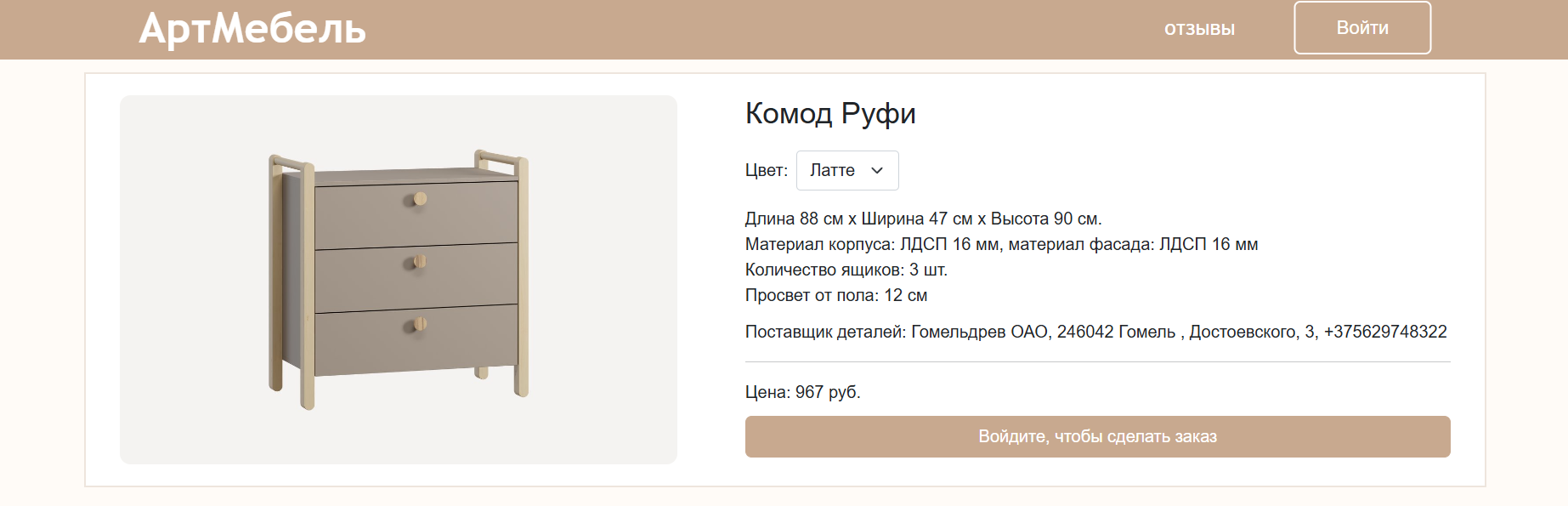


Рисунок 3.5 – Страница с подробностями товара для незарегистрированного пользователя

При нажатии на кнопку «Под заказ» на главной странице пользователь переходит на страницу с подробностями товара с параметрами. Здесь доступна кнопка «Войдите, чтобы сделать заказ», при нажатии на которую пользователя перебросит на страницу авторизации.

На рисунке 3.6 представлена Страница с подробностями товара с параметрами для незарегистрированного пользователя.

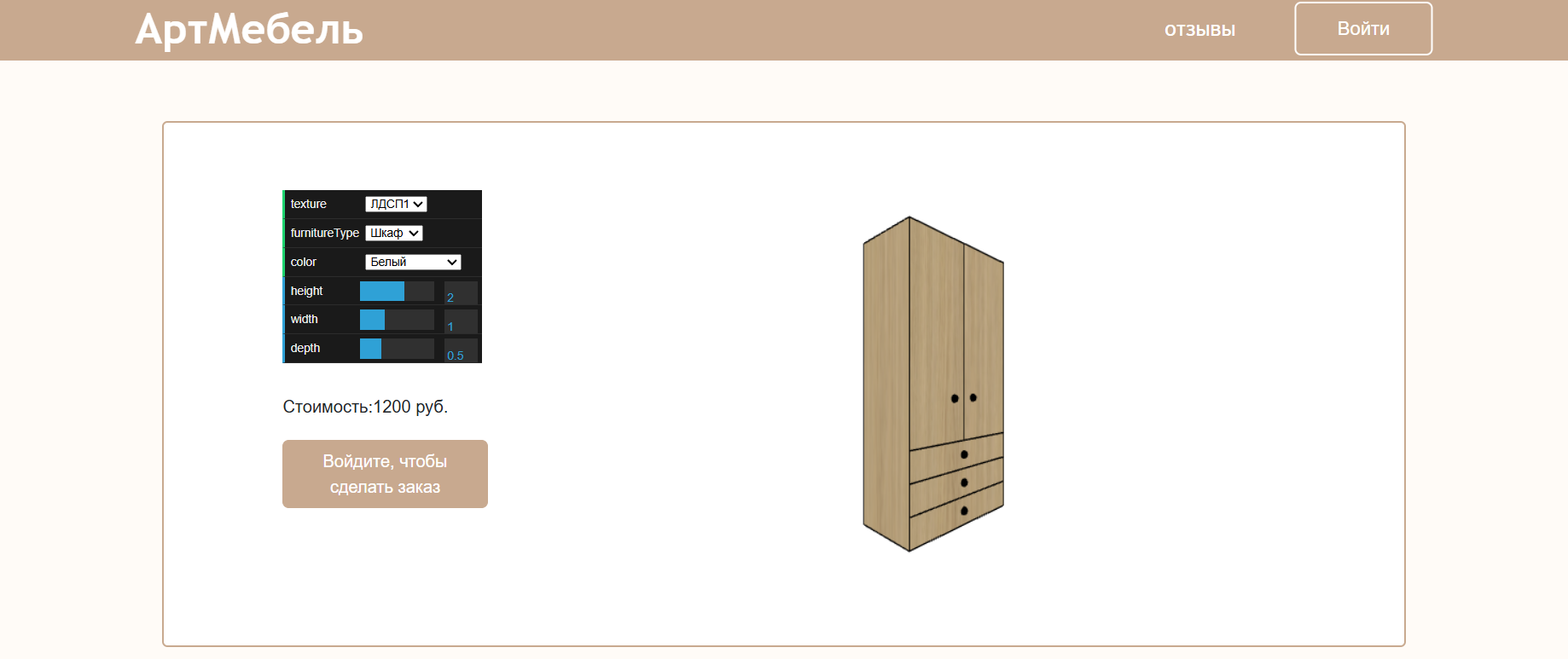


Рисунок 3.6 – Страница с подробностями товара с параметрами для незарегистрированного пользователя

При нажатии на кнопку «отзывы» пользователь перейдёт на страницу с отзывами.

На рисунке 3.7 представлена страница с отзывами для незарегистрированного пользователя.

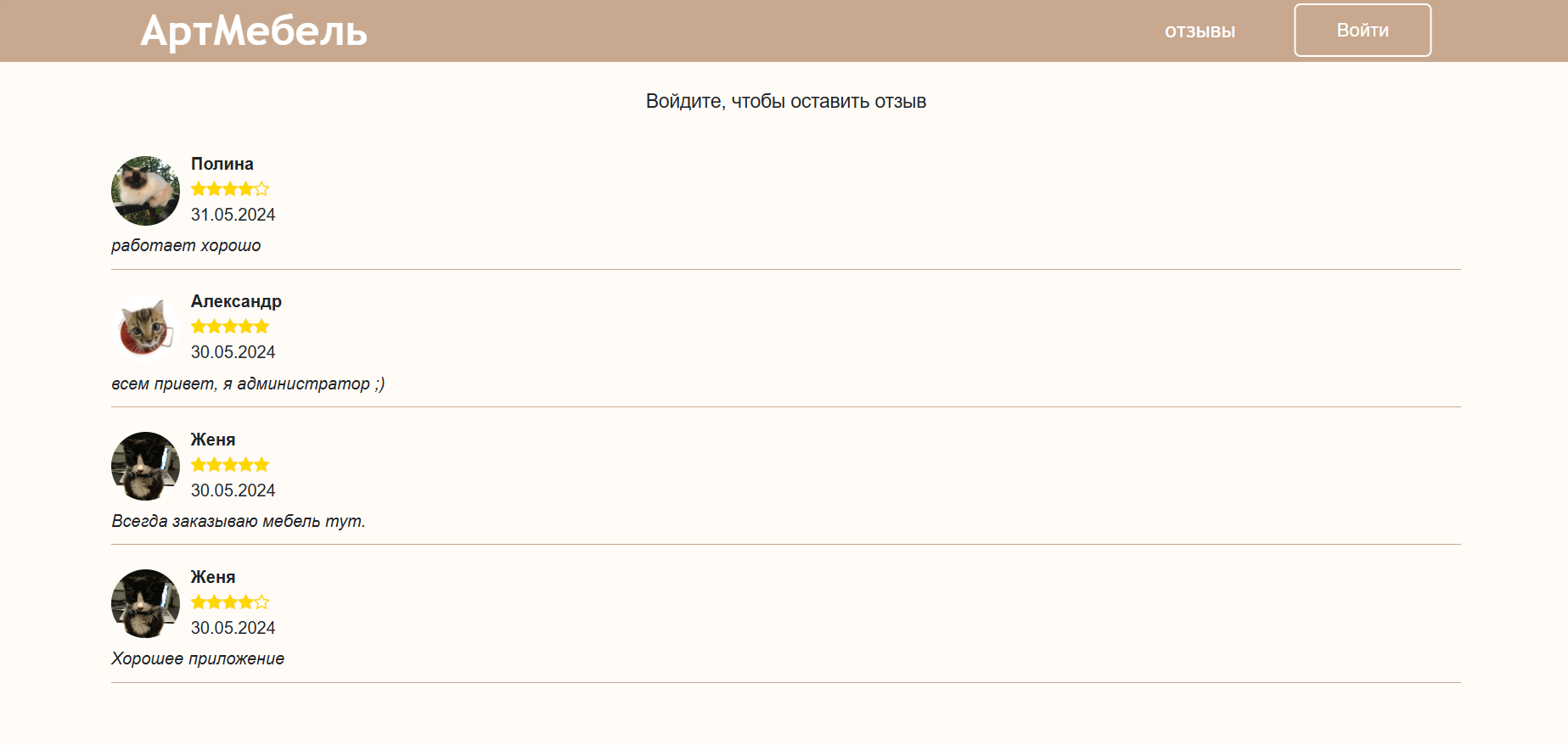


Рисунок 3.7 – Страница с отзывами для незарегистрированного пользователя

При нажатии на кнопку «Войти» незарегистрированный пользователь перейдёт на страницу авторизации.

На рисунке 3.8 представлена форма авторизации пользователя.

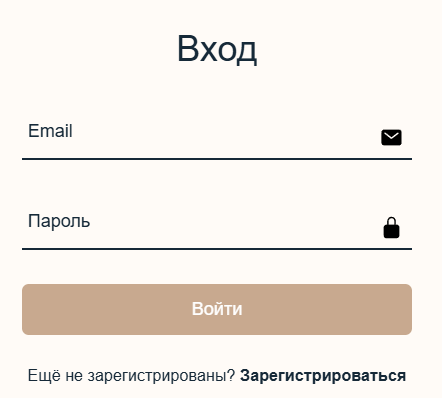


Рисунок 3.8 – Форма авторизации пользователя

Если пользователь ещё не зарегистрирован, он может нажать на ссылку «Зарегистрироваться» и он перейдёт на страницу регистрации.

На рисунке 3.9 представлена форма регистрации пользователя.

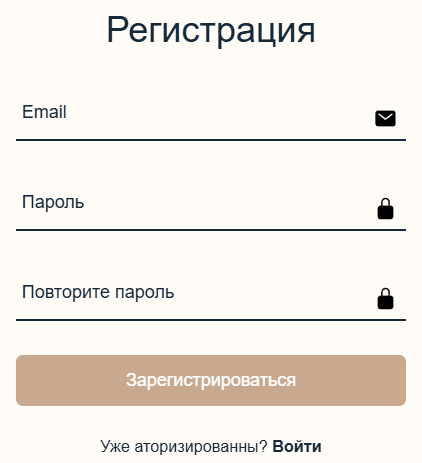


Рисунок 3.9 – Форма регистрации пользователя

После того, как пользователь авторизировался, пользователя перекидывает на главную страницу с каталогом мебели. На рисунке 3.10 представлена главная страница с каталогом мебели для зарегистрированного пользователя.

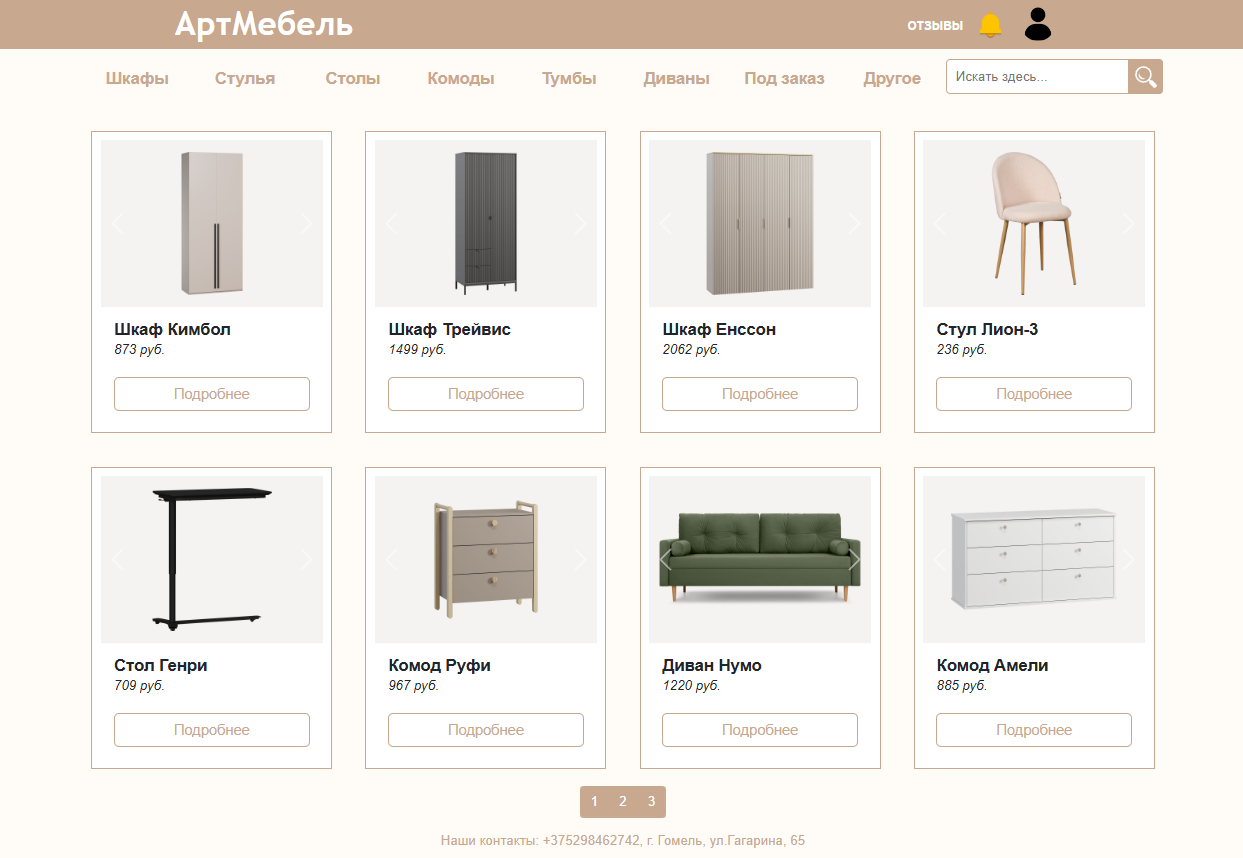


Рисунок 3.10 – Главная страница для зарегистрированного пользователя

При нажатии на иконку пользователя открывается меню. В нем доступны кнопки «Профиль», «Мои заказы», «Выйти». На рисунке 3.11 представлено выпадающее меню при нажатии на иконку зарегистрированного пользователя.

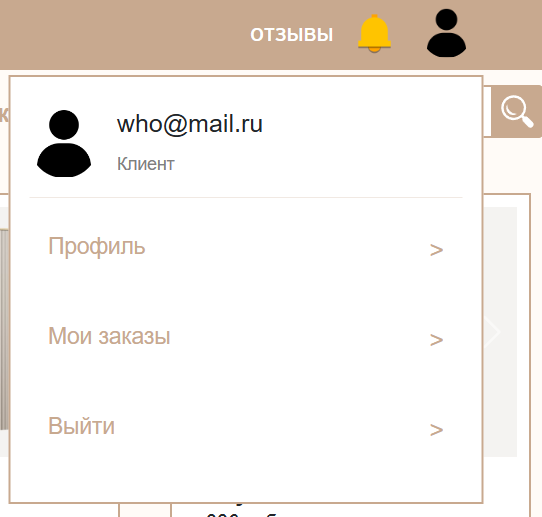


Рисунок 3.11 – Выпадающее меню при нажатии на иконку зарегистрированного пользователя

При нажатии на кнопку «Профиль» в выпадающем меню пользователь переходит на страницу со своим профилем, где он может редактировать свою информацию. На странице 3.12 представлена страница с профилем зарегистрированного пользователя.

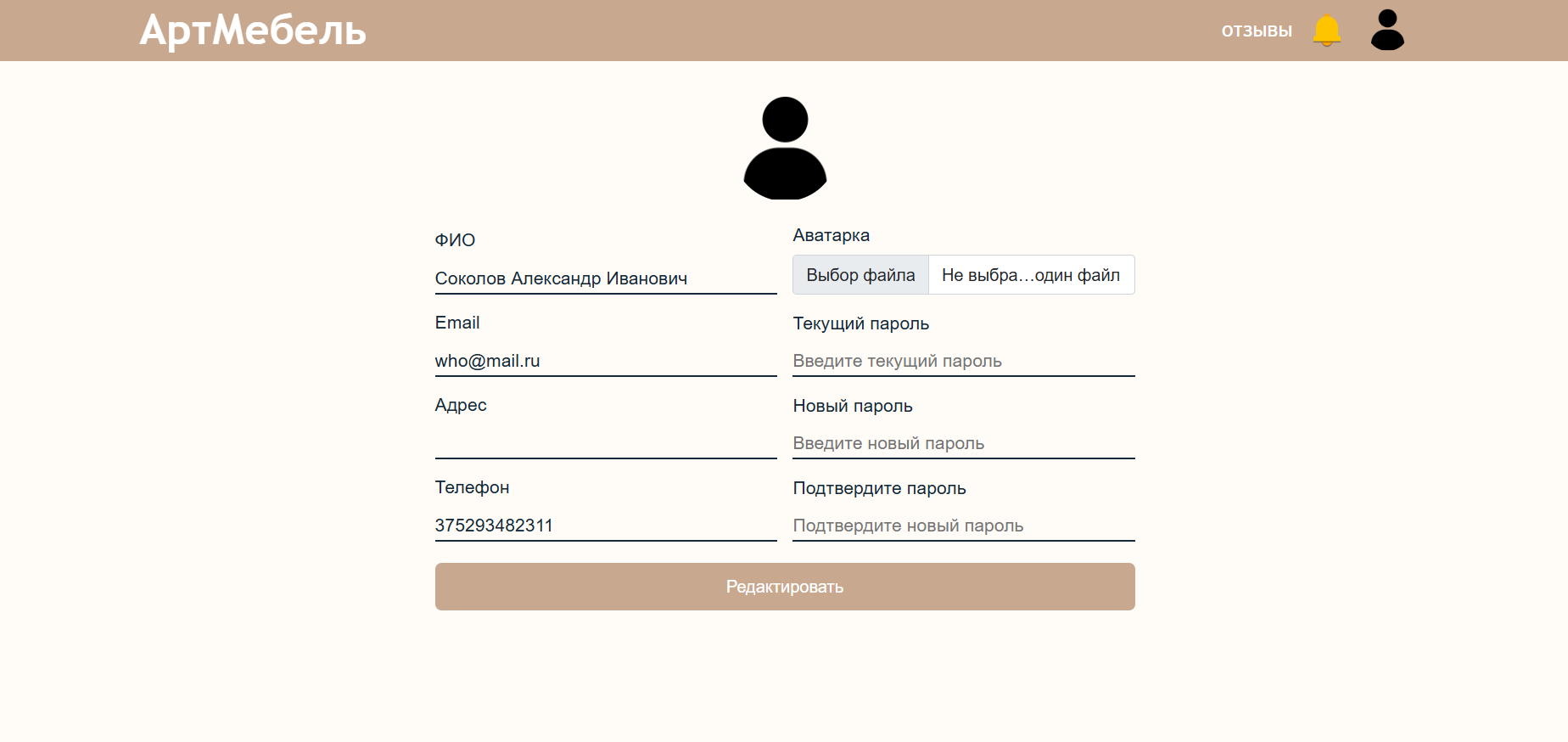


Рисунок 3.12 – Страница с профилем зарегистрированного пользователя

Пользователь вводит данные, которые он хочет отредактировать, вводит свой текущий пароль, и если он совпадает, то данные обновляются. На рисунке 3.13 представлена страница с профилем зарегистрированного пользователя после редактирования.

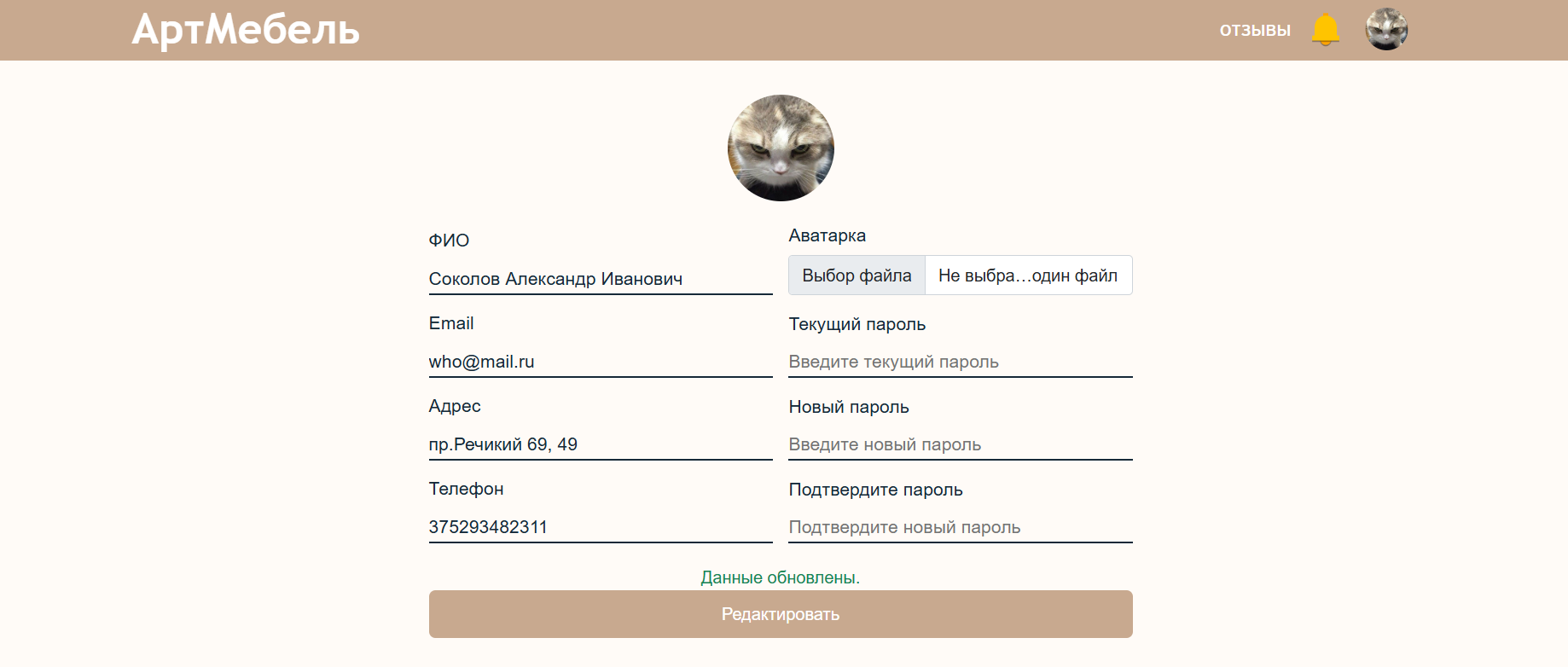


Рисунок 3.13 – Страница с профилем зарегистрированного пользователя после редактирования

При нажатии на кнопку «Мои заказы» в выпадающем меню пользователь переходит на страницу со своими заказами. На нем доступны кнопки «Отменить» и «К товару». Кнопка «Отменить» доступна только если заказ ещё не в производстве. При нажатии на кнопку «Отменить» пользователь отменит заказ. При нажатии на кнопку «К товару» пользователь перейдёт на страницу с подробностями товара, который он заказал. На рисунке 3.14 показана страница с заказами зарегистрированного пользователя.



Рисунок 3.14 – Страница с заказами зарегистрированного пользователя

При нажатии на кнопку «Подробнее» на главной странице пользователь перейдет на страницу с подробностями товара. На ней доступен выбор цвета мебели, описание мебели, цена, а также кнопка «Заказать».

На рисунке 3.15 показана страница с подробностями товара для зарегистрированного пользователя.



Рисунок 3.15 – Страница с подробностями товара для зарегистрированного пользователя

При нажатии на кнопку «Заказать» на странице с подробностями и товаре, пользователю открывается модальное окно с подтверждением заказа. На рисунке 3.16 показана модальное окно с подтверждением заказа товара для зарегистрированного пользователя.

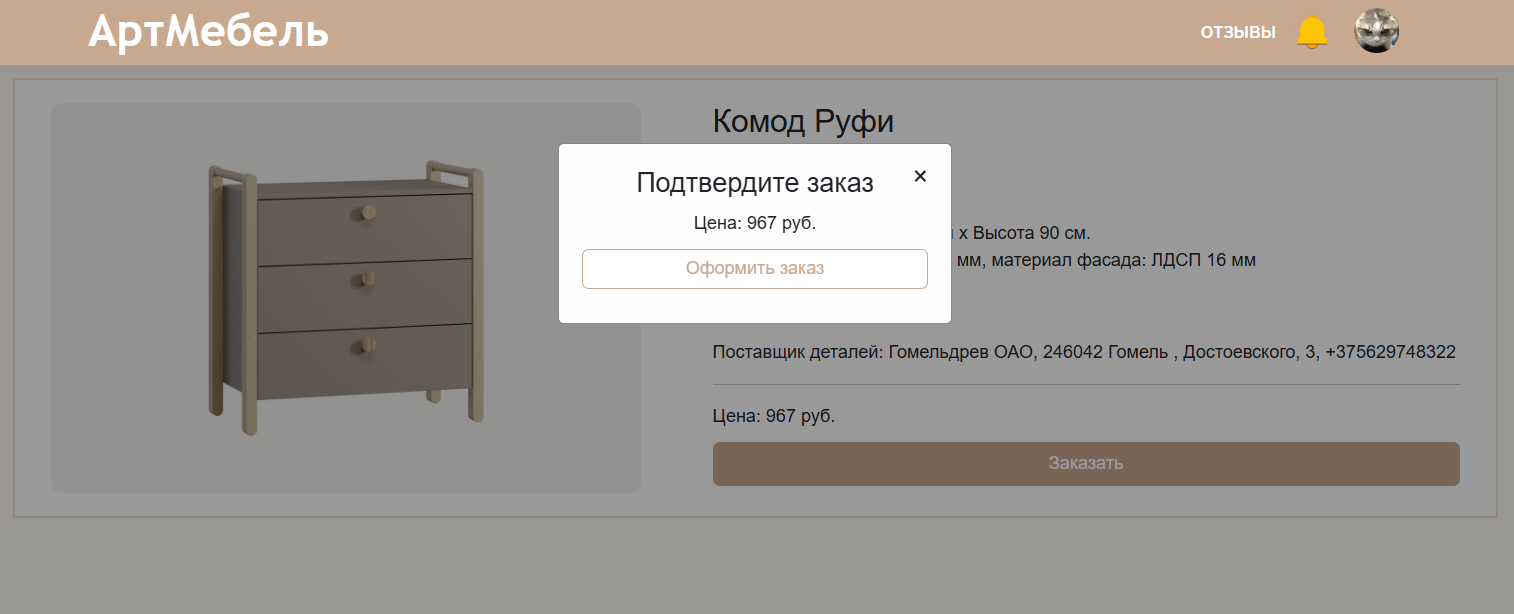


Рисунок 3.16 – Модальное окно с подтверждением заказа товара для зарегистрированного пользователя

При нажатии на кнопку «Под заказ» на главной странице пользователь перейдет на страницу с *3D*-моделями мебели. Здесь он может задать параметры мебели и посмотреть, как она будет выглядеть и заказать мебель с выбранными параметрами. На рисунке 3.17 показана страница с *3D*-моделью мебели.

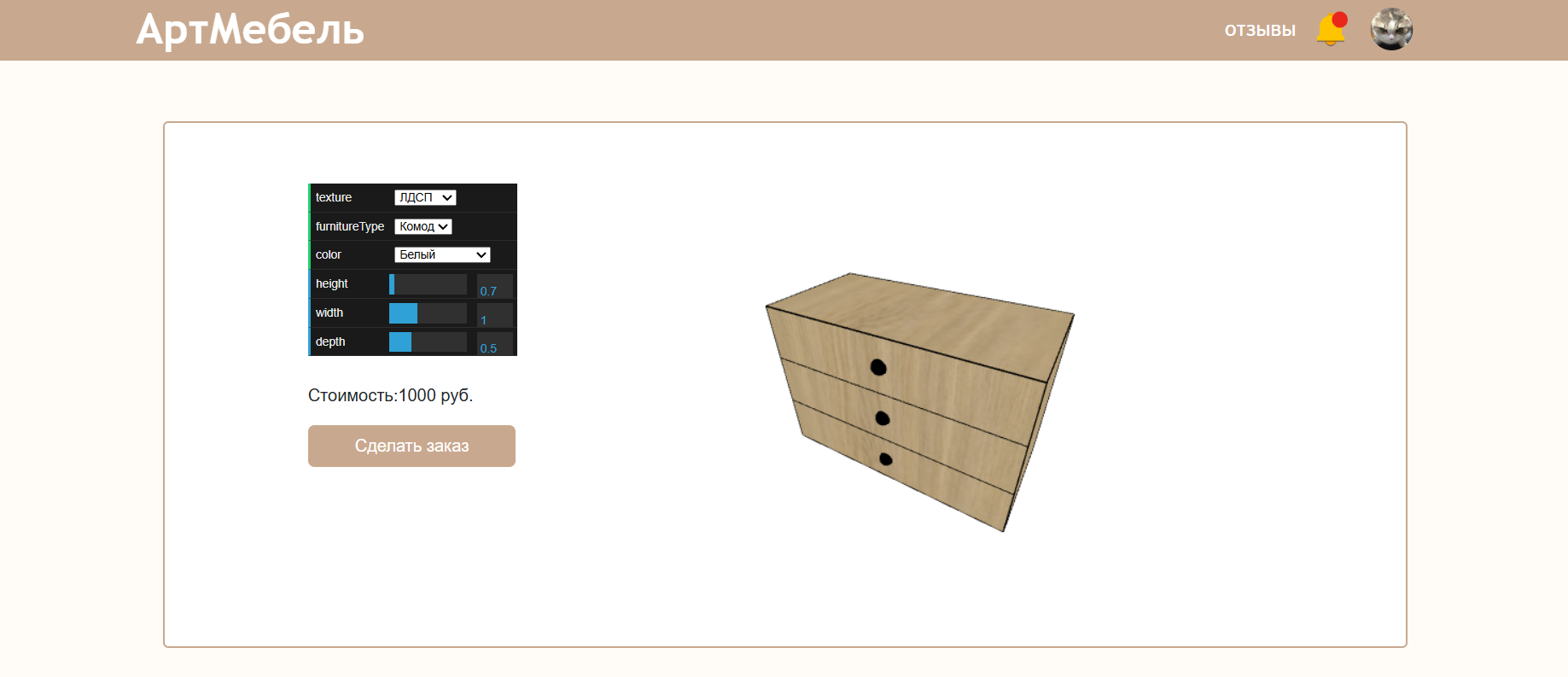


Рисунок 3.17 – Страница с *3D*-моделью мебели

При нажатии на кнопку «Сделать заказ» появляется модальное окно с подтверждением заказа товара с параметрами для зарегистрированного пользователя. На рисунке 3.18 показана модальное окно с подтверждением заказа мебели с параметрами для зарегистрированного пользователя.

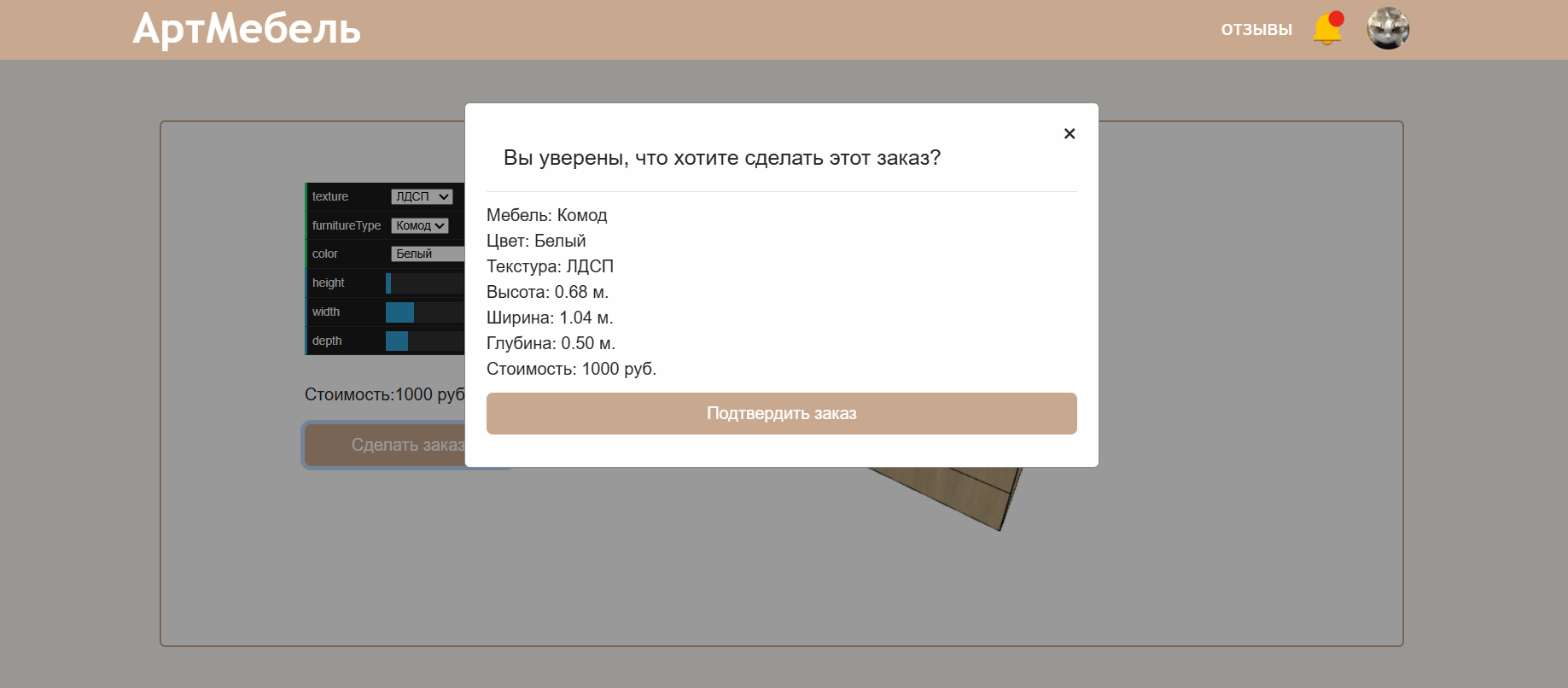


Рисунок 3.18 – Модальное окно с подтверждением заказа мебели с параметрами для зарегистрированного пользователя

После того, как пользователь заказал товар, или при обновлении статус заказа, или когда заказ готов и его можно забирать, ему приходит уведомления, которые он может посмотреть и пометить как прочитанные. На рисунке 3.19 представлено выпадающее окно с уведомлениями для клиента.

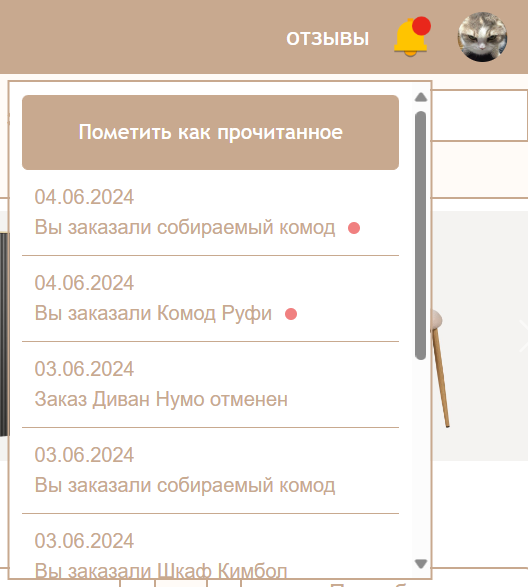


Рисунок 3.19 – Выпадающее окно с уведомления для клиента

В этом окне доступна кнопка «Пометить как прочитанное», при нажатии на которую, уведомления помечаются как прочитанные. На рисунке 3.20 представлено выпадающее окно с уведомлениями для клиента, после нажатия на кнопку «Пометить как прочитанное».

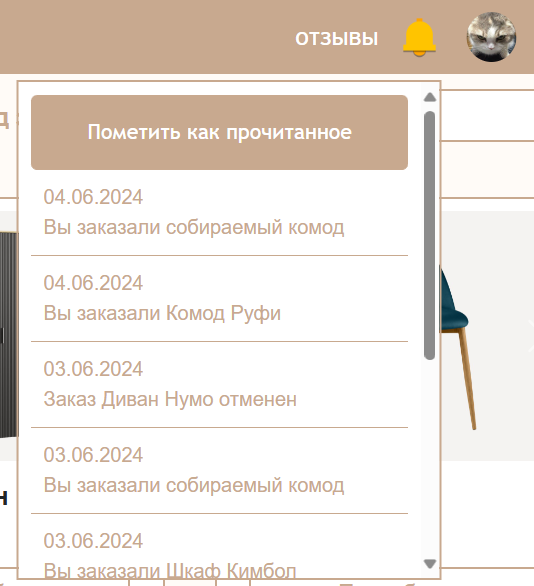


Рисунок 3.20 – Выпадающее окно с уведомления для клиента

При нажатии на кнопку «отзывы» на шапке страницы, пользователь переходит на страницу с отзывами, где он может посмотреть все отзывы на приложение и добавить свой отзыв. На рисунке 3.21 представлена страница с отзывами клиентов.

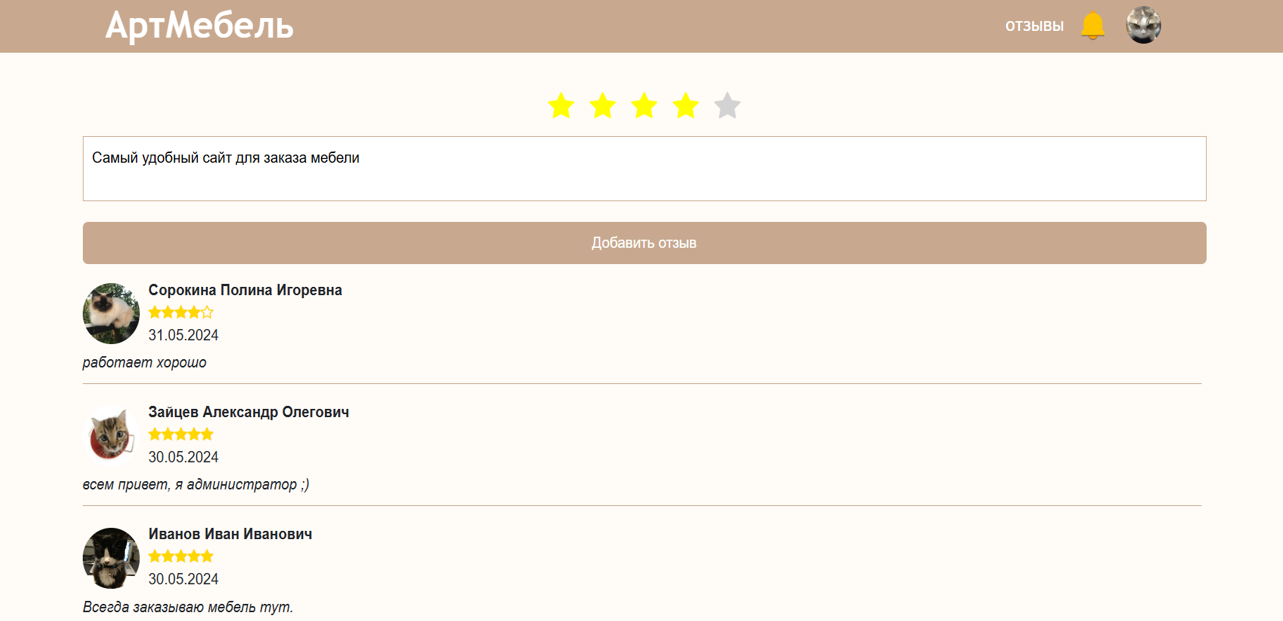


Рисунок 3.21 – Страница с отзывами клиентов

При нажатии на кнопку «Добавить отзыв», пользователь оставит свой отзыв, если он написал отзыв и поставил оценку. На странице 3.22 представлена страница с отзывами после добавления нового отзыва.

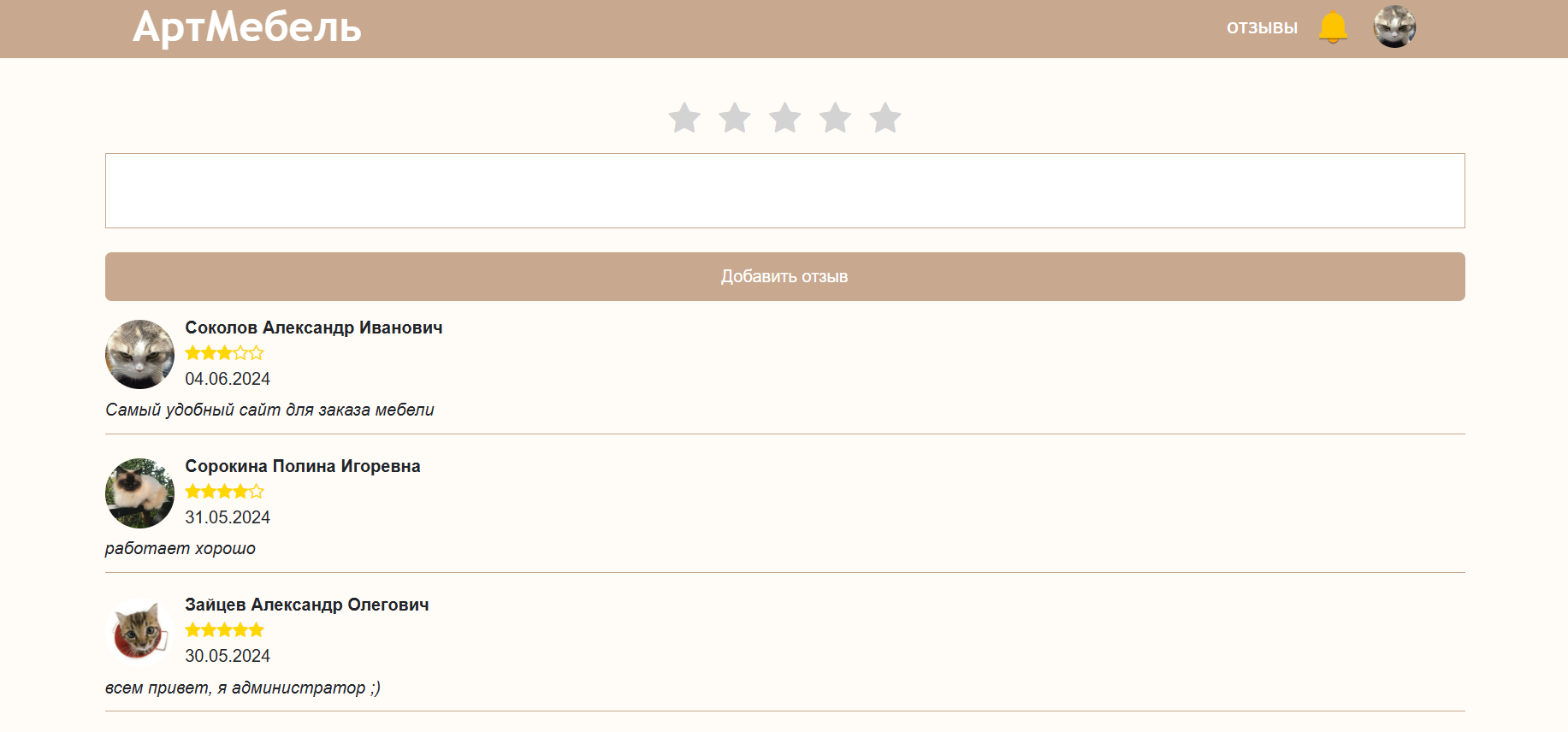


Рисунок 3.22 – Страница с отзывами клиентов после добавления нового отзыва

На рисунке 3.23 показана шапка страницы для роли «Специалист по приему заказов». Специалисту по приему заказов доступен переход на страницы «управление заказами» и «мониторинг».

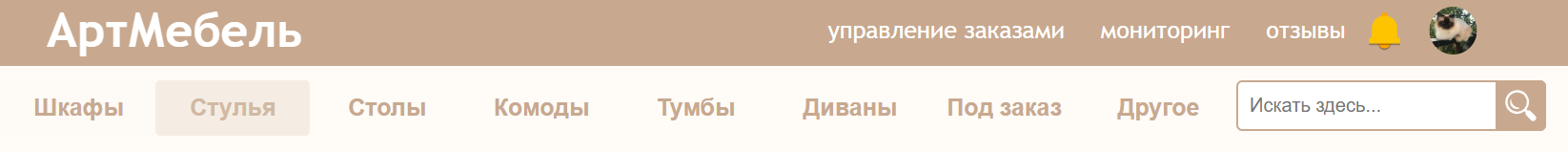


Рисунок 3.23 – Шапка страницы для специалиста по приему заказов

При нажатии на вкладку «управление заказами», специалист переходит на страницу с заказами. Здесь специалист может смотреть заказы всех клиентов, либо определенного клиента, которого специалист может выбрать из списка. Выбрав заказ клиента, он может обновить статус заказа, для этого специалист должен выбрать статус и нажать кнопку «Обновить». На рисунке 3.24 представлена страница с заказами для специалиста по приему заказов.

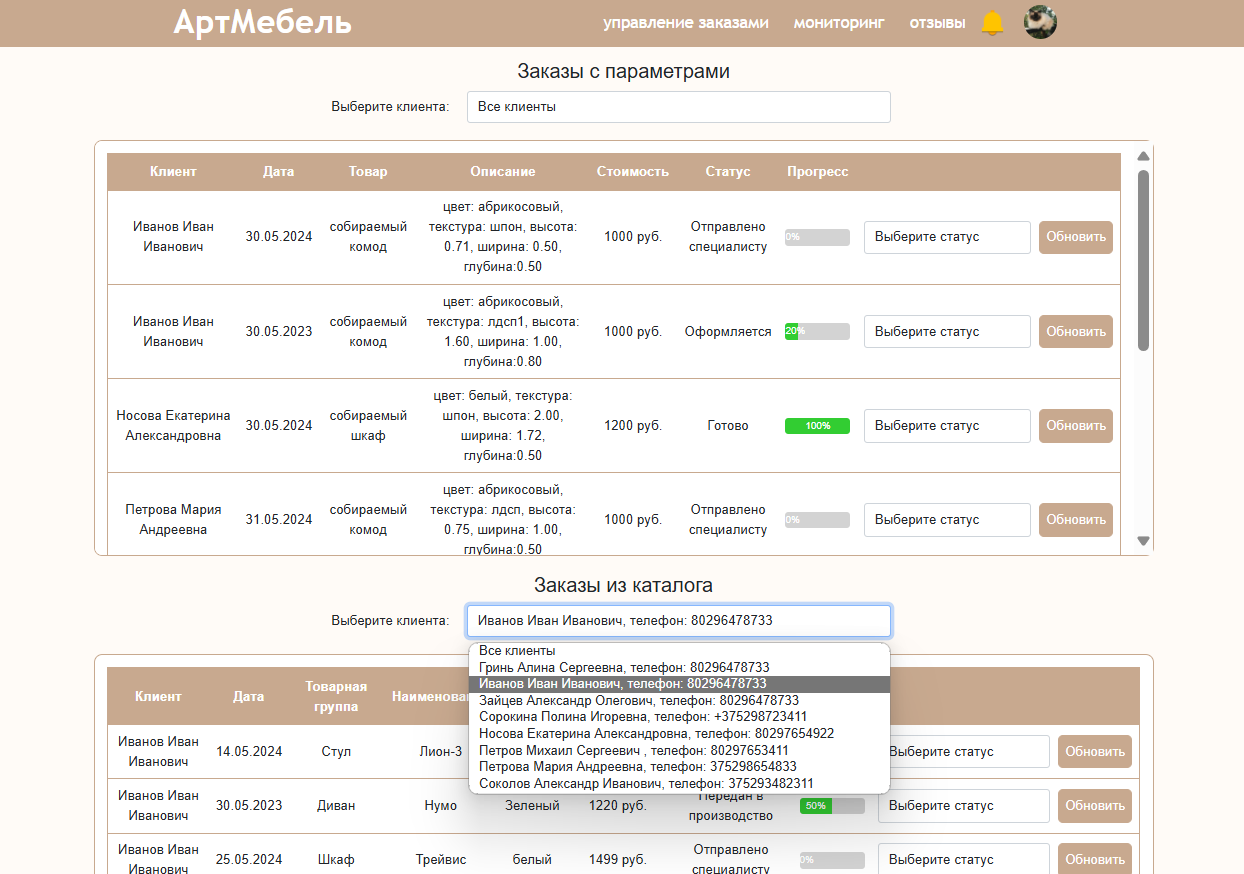


Рисунок 3.24 – Страница с заказами для специалиста по приему заказов

При нажатии на вкладку «мониторинг», специалист переходит на страницу со статистикой. Специалист задает промежуток времени, за который были сделаны заказы, нажимает на «Применить», и ему выводится диаграмма с количеством заказанных товаров по категориям, а также ниже список заказов, заказы на которые были отменены. Также доступен вывод самых популярных по заказам или же самых непопулярных по заказам товаров. Специалист может указать сколько самых заказываемых товаров он хочет вывести. Все данные со статистикой специалист может скачать себе на устройство. На рисунке 3.25 представлена страница со статистикой по заказам на заданный промежуток времени.

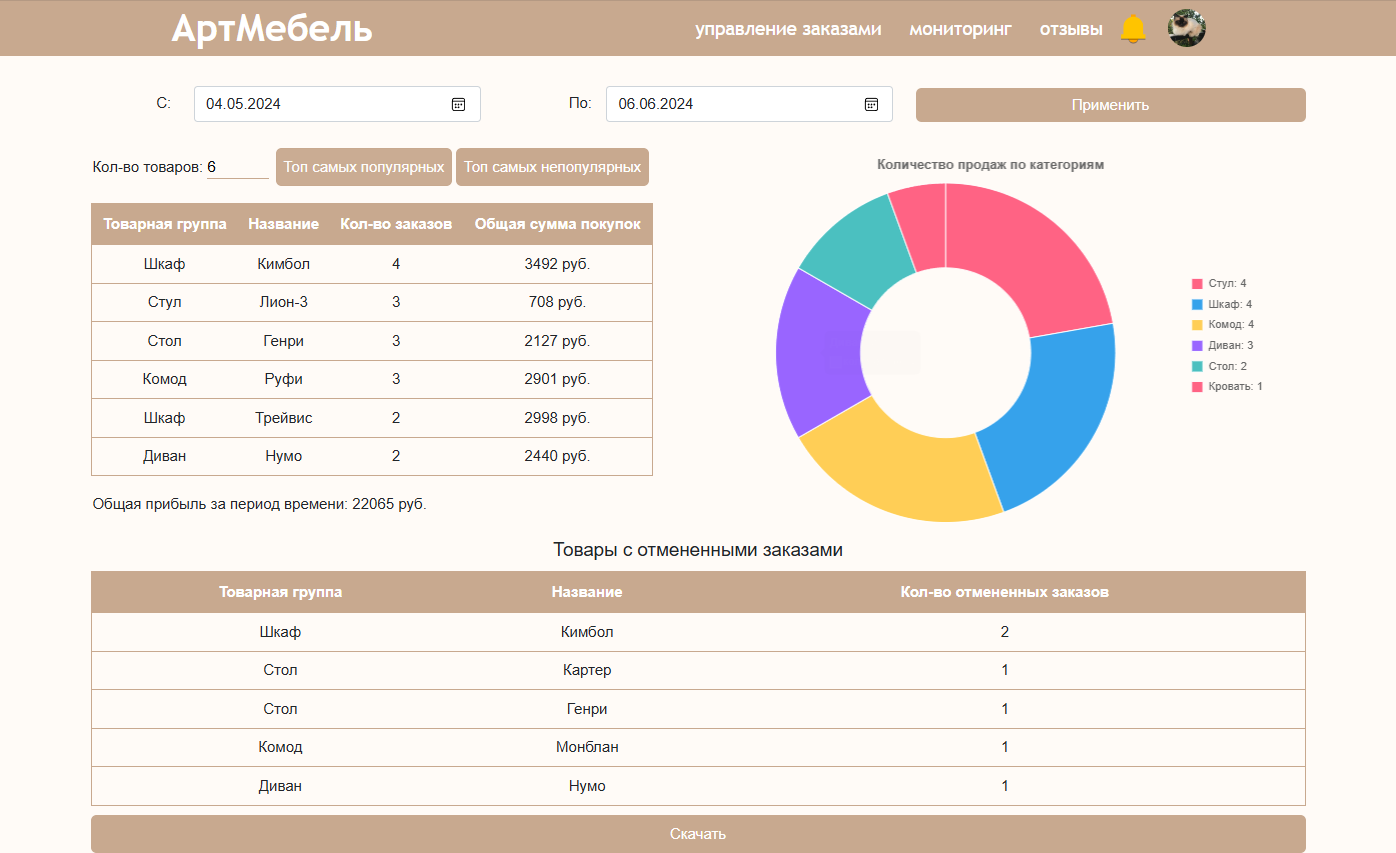


Рисунок 3.25 – Страница с статистикой по заказам на заданный промежуток времени

На странице со статистикой также доступна кнопка «Скачать». При нажатии на кнопку «Скачать» скачивается *pdf*-файл со статистикой. На рисунке 3.26 показано содержимое скачанного *pdf*-файла со статистикой.



Рисунок 3.26 – Содержимое скачанного *pdf*-файла со статистикой

На рисунке 3.27 представлена шапка страницы для менеджера по управлению запасами. Менеджеру по управлению запасами доступен весь функционал зарегистрированного пользователя, а также ему доступна вкладка «управление запасами».

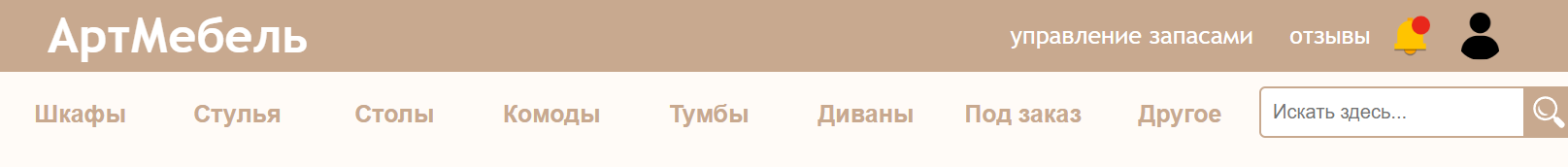


Рисунок 3.27 – Шапка страницы для менеджера по управлению запасами

При нажатии на вкладку «управление запасами», менеджер переходит на страницу с запасами. Здесь он может выбрать склад, смотреть какие товары находятся на выбранном складе, редактировать товары, удалять, добавлять новые товары на склад, добавлять на склад уже существующие товары, но с другим материалом, цветом.

На рисунке 3.28 представлена страница управления запасами.

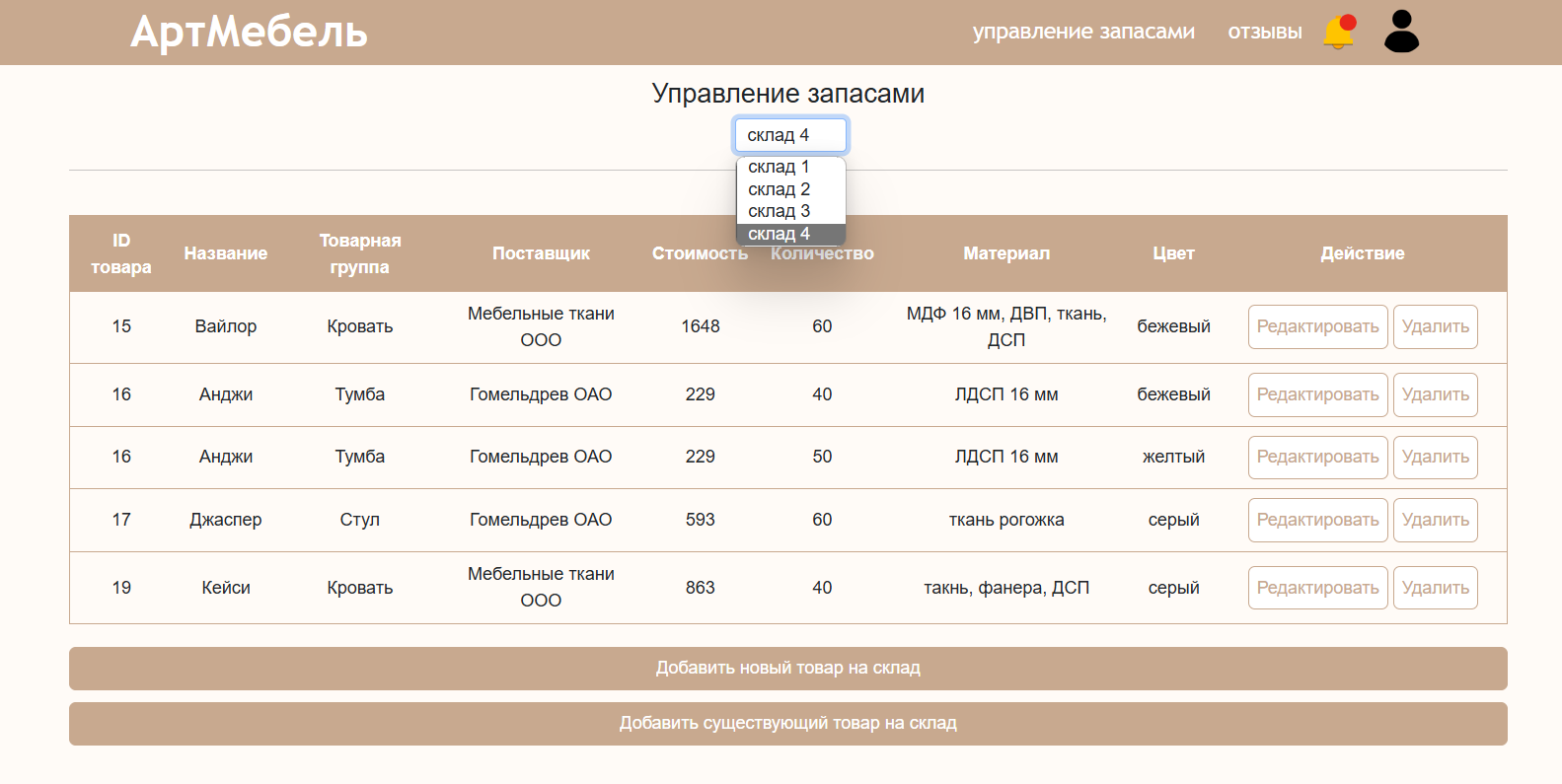


Рисунок 3.28 – Страница для управления запасами

При нажатии на кнопку «Добавить новый товар на склад» открывается модальное окно с добавление товара. Внутри модального окна доступна кнопка «Добавить товар на склад», которую менеджер может нажать после того, как заполнит поля необходимой информацией, такой как название, стоимость, цвет, материал, количество, описание, номер склада, товарная группа, поставщик деталей, файл с картинкой.

На рисунке 3.29 представлено модальное окно с добавлением нового товара на склад.

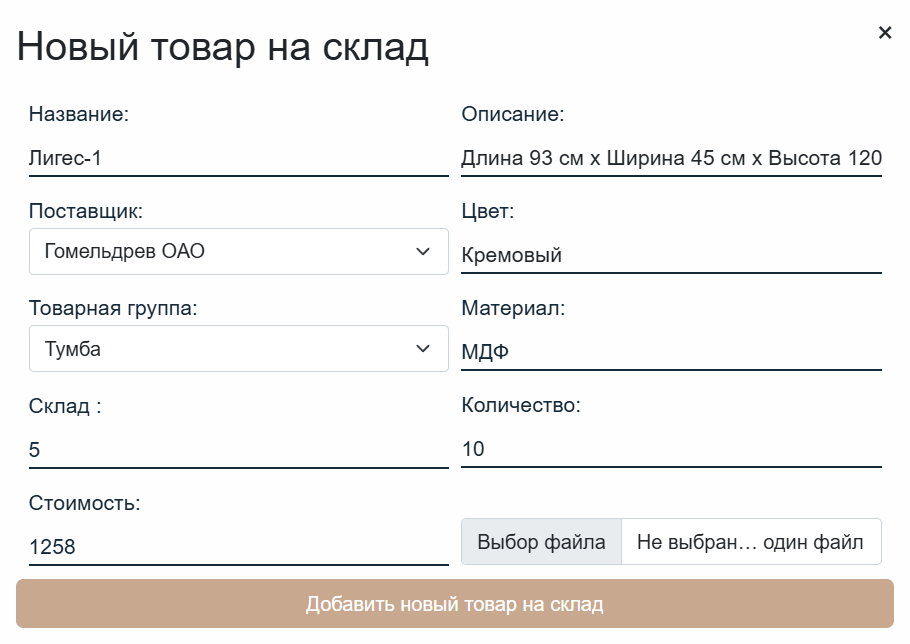


Рисунок 3.29 – Модальное окно с добавлением нового товара

При нажатии на кнопку «Добавить существующий товар на склад» менеджеру показывается модальное окно с добавлением уже существующего товара, но с другими характеристиками.

На рисунке 3.30 представлено модальное окно добавления существующего товара.

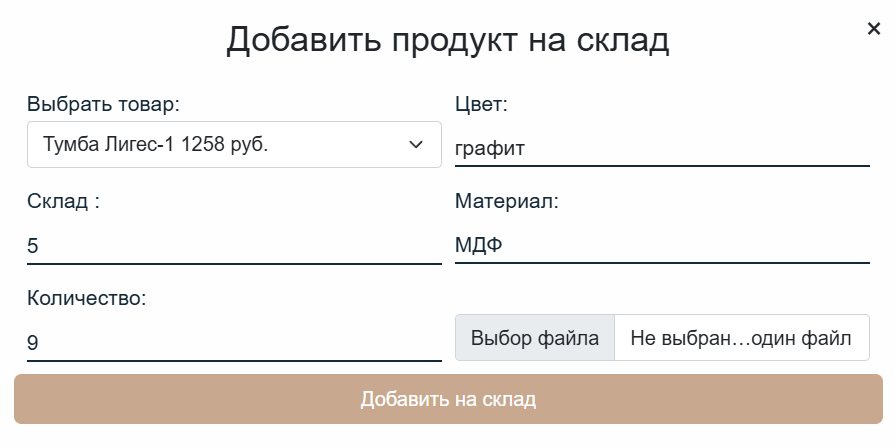


Рисунок 3.30 – Модальное окно добавления существующего товара

При нажатии на «Редактировать» менеджеру показывается модальное окно с редактированием товара.

На рисунке 3.31 представлено модальное окно с редактированием товара.

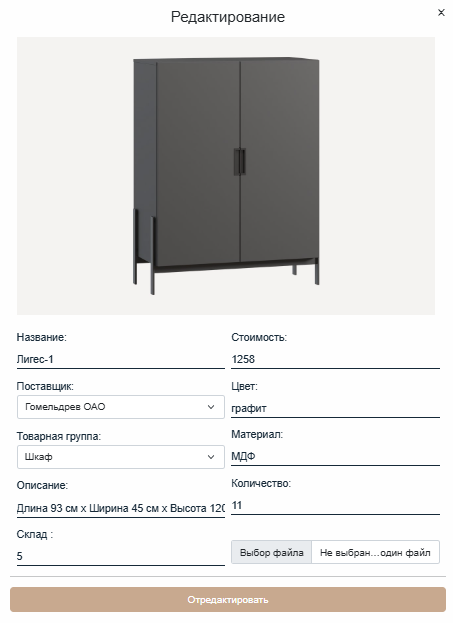


Рисунок 3.31 – Модальное окно с редактированием товара

При нажатии на «Удалить» менеджеру показывается модальное окно с подтверждением удаления. На рисунке 3.32 показано модальное окно с подтверждением удаления товара с определенными характеристиками со склада.

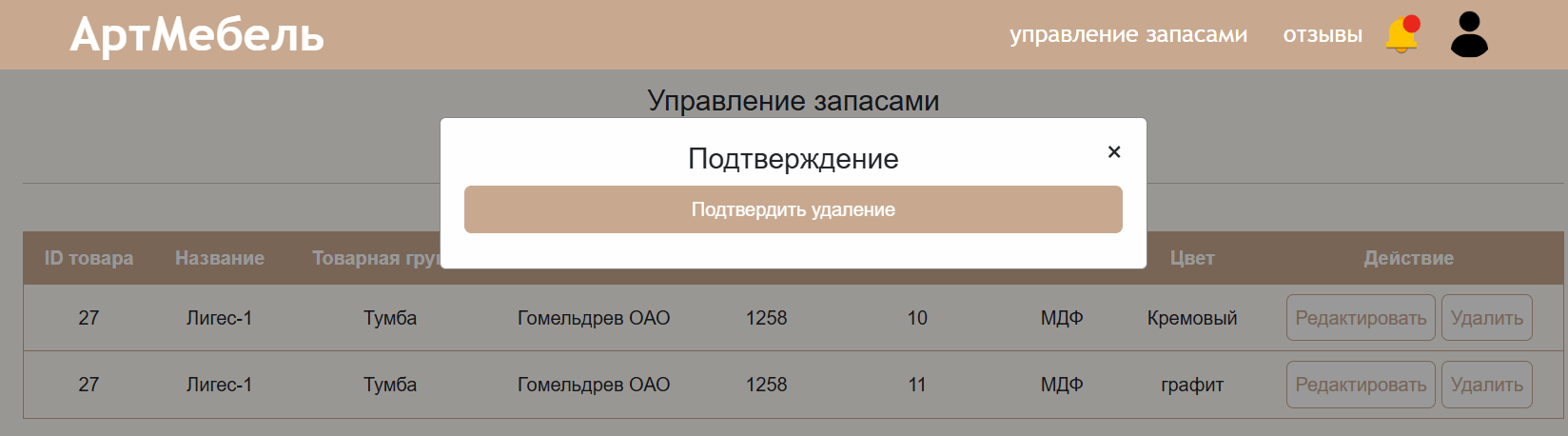


Рисунок 3.32 – Модальное окно с удалением товара со склада

Администратору доступна вкладка «управление пользователями» и весь функционал зарегистрированного пользователя. Администратору предоставлено назначение ролей пользователям и добавление ролей.

Шапка страницы для администратора представлена на рисунке 3.33.

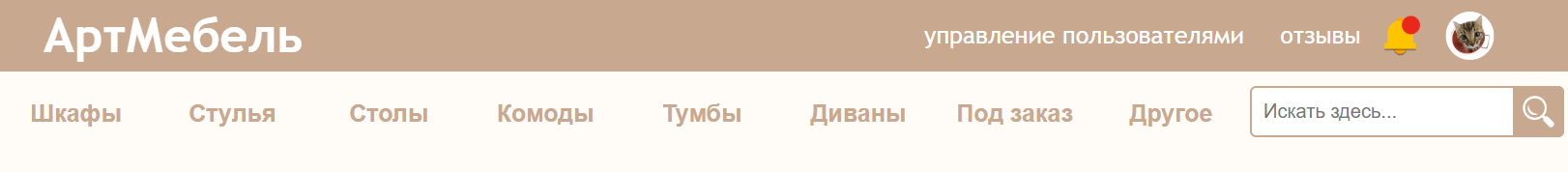


Рисунок 3.33 – Шапка страницы для администратора

При нажатии на вкладку «управление пользователями», администратор переход на страницу со списком ролей. На рисунке 3.34 представлена страница со списком ролей.

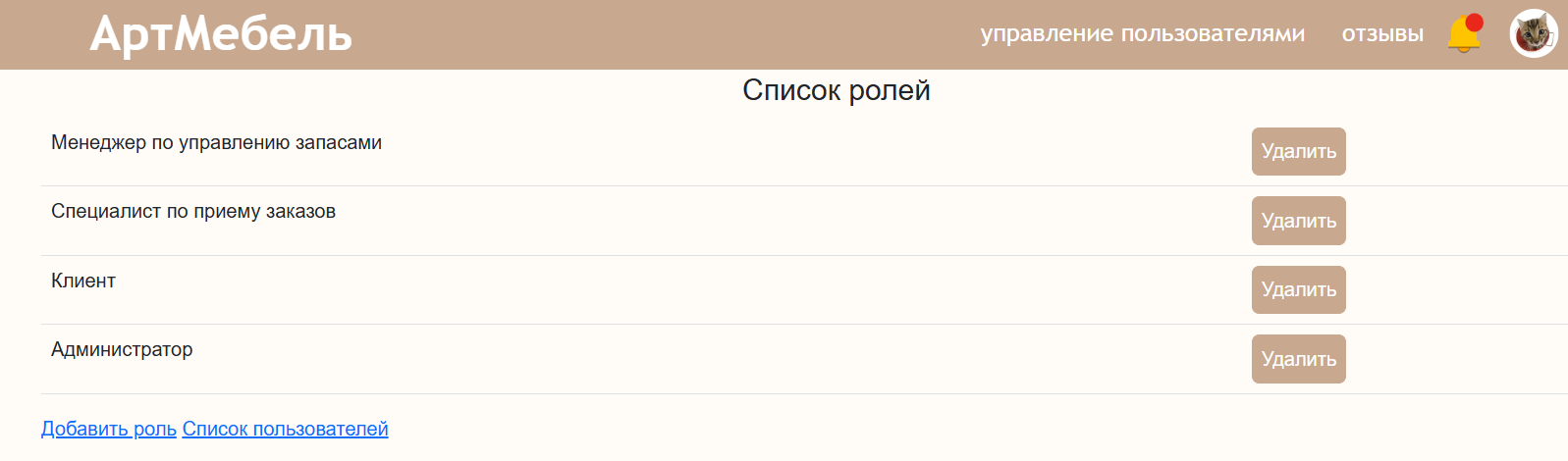


Рисунок 3.34 – Страница со списком ролей

При нажатии на ссылку «Добавить роль», администратора перейдет на страницу с добавлением роли. После ввода названия роли и нажатия на кнопку «Добавить» он сможет добавить новую роль. На рисунке 3.35 представлена страница с добавлением роли.

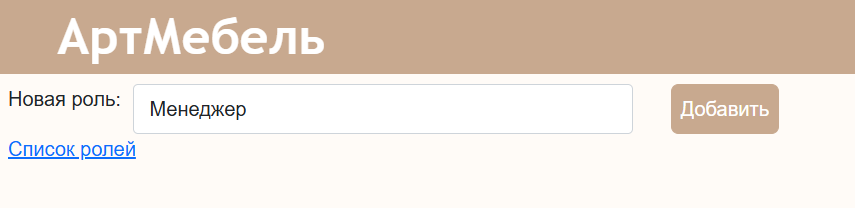


Рисунок 3.35 – Страница с добавлением роли

При нажатии на ссылку «Список пользователей», администратора перейдет на страницу со списком пользователей.

На рисунке 3.36 представлена страница со списком пользователей.

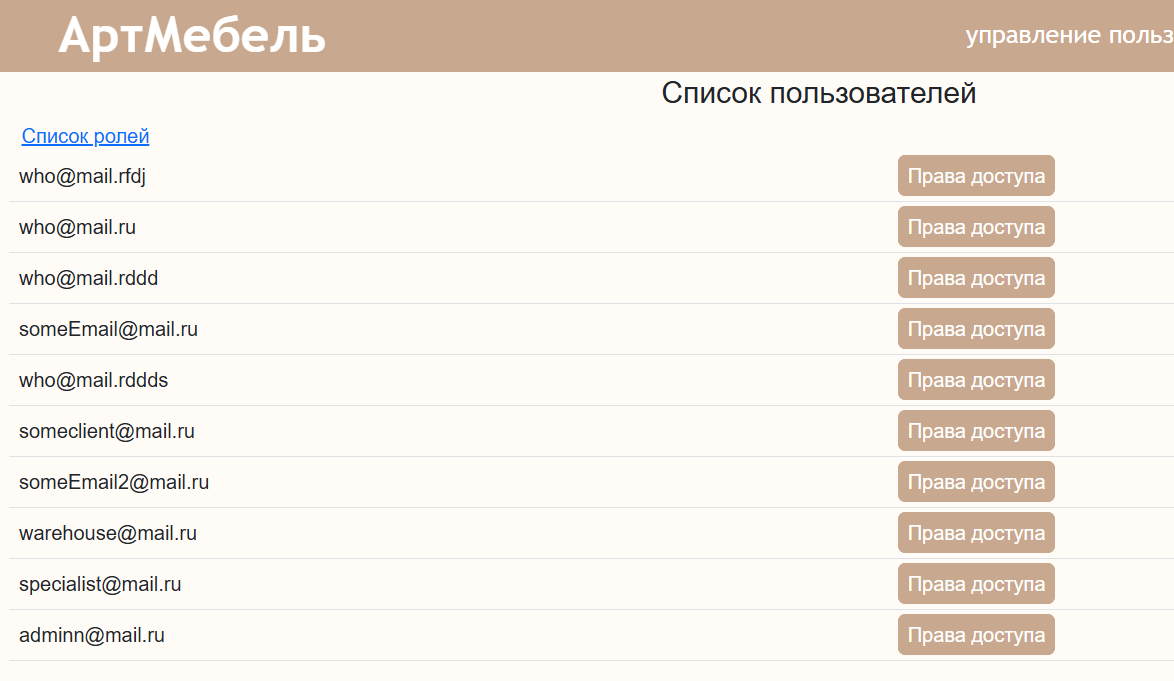


Рисунок 3.36 – Страница со списком пользователей

При нажатии на кнопку «Права доступа» на странице со списком клиентов, администратор переходит на страницу, где он может назначить роль пользователю и сохранить. На рисунке 3.37 страница представлена страница с правами доступа пользователя.

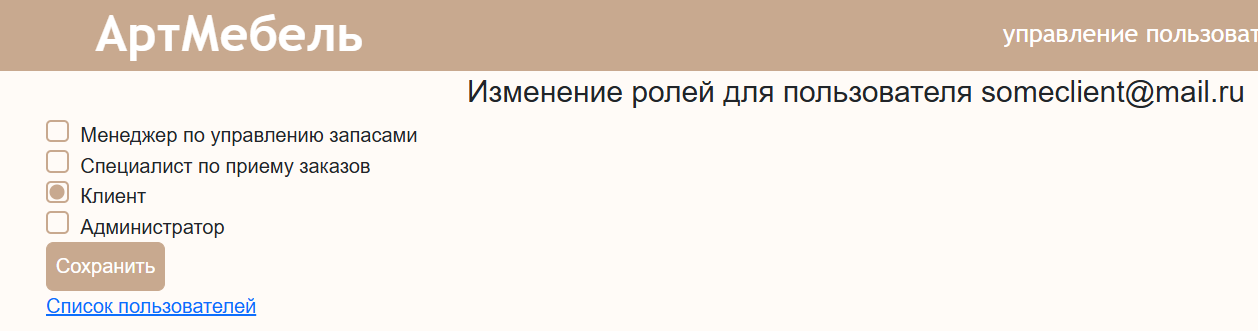


Рисунок 3.37 – Страница с правами доступа пользователя

# **ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **Ручное тестирование программного комплекса**

Ручное тестирование программного комплекса является процессом проверки функциональности, качества и удобства использования системы путем ручного взаимодействия с ее компонентами. Этот процесс позволяет выявить потенциальные проблемы и ошибки, которые могут возникнуть при использовании приложения.

Тестирование в различных сценариях также позволяет убедиться, что все функции и компоненты приложения работают должным образом и соответствуют ожиданиям пользователей. Например, можно протестировать процесс оформления заказа, процесс регистрации и входа в систему, процесс обновления данных в профиле и другое.

Тестирование является неотъемлемой частью создания приложения, так как необходимо проверить приложение в различных сценариях. Гостю недоступен функционал зарегистрированного пользователя, то есть недоступна кнопка «Заказать», кнопка «Оставить отзыв», оповещения, личный кабинет пользователя, заказы пользователя.

Если при регистрации пользователь не заполнит поля, появится надпись «Заполните это поле». На рисунке 4.1 показана форма регистрации с пустыми полями.

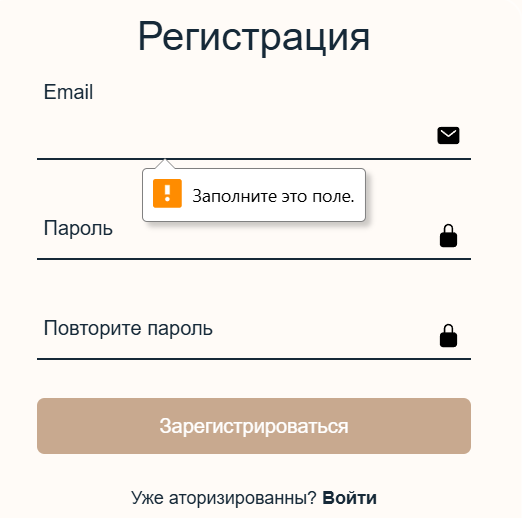


Рисунок 4.1 – Форма регистрации с пустыми полями

Если при регистрации пользователь введёт почту, которая уже зарегистрирована, пользователь будет выдано сообщение об ошибке.

На рисунке 4.2 показана форма регистрации при вводе уже существующей почты.

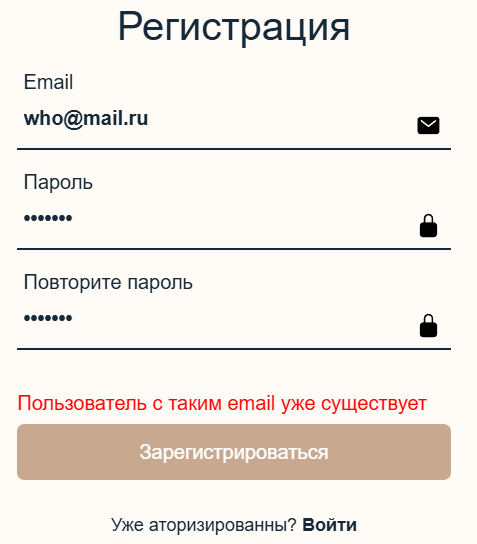


Рисунок 4.2 – Форма регистрации при вводе уже существующей почты

Если при регистрации пользователь введет неправильный формат почты, пользователь не сможет зарегистрироваться и будет выдано сообщение об ошибке. На рисунке 4.3 показана форма регистрации с вводом почты неправильного формата.

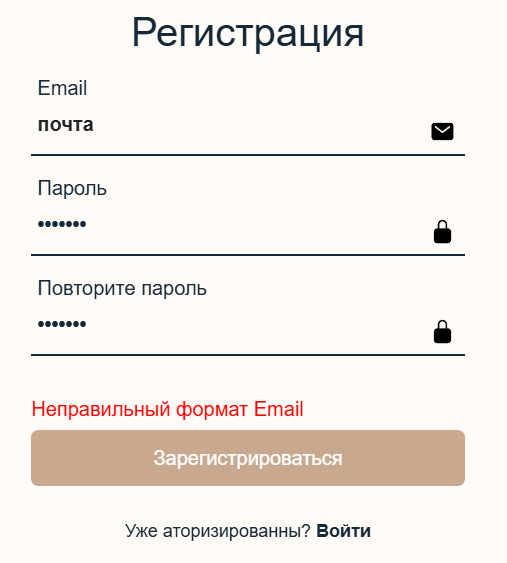


Рисунок 4.3 – Форма регистрации с вводом почты неправильного формата

Если при регистрации пользователь неправильно повторит пароль, пользователь не сможет зарегистрироваться и будет выдано сообщение об ошибке. На рисунке 4.4 показана форма регистрации, если пользователь неправильно введет пароль, который нужно повторить.

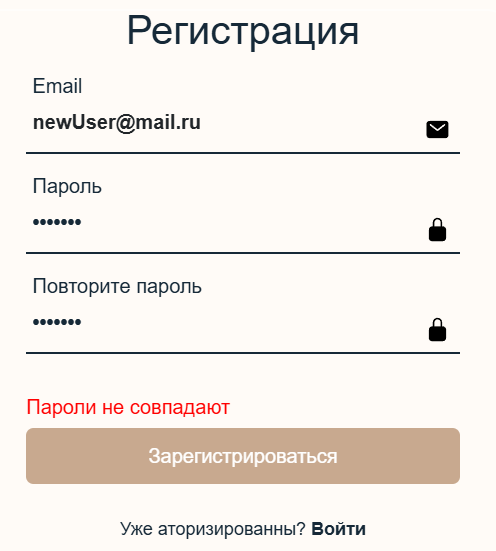


Рисунок 4.4 – Форма регистрации, если пользователь неправильно введет пароль, который нужно повторить

Если при авторизации пользователь не заполнит поля, появится надпись «Заполните это поле». На рисунке 4.5 показана форма авторизации с пустыми полями.

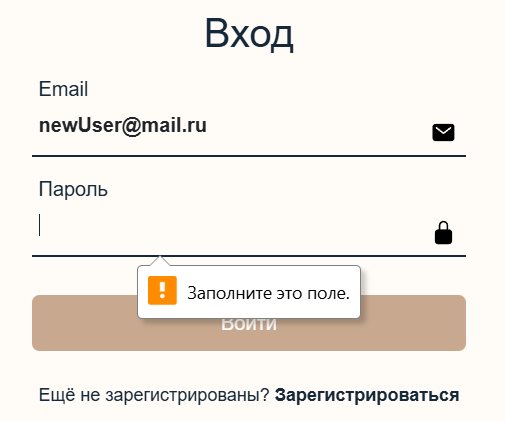


Рисунок 4.5 – Форма авторизации с пустыми полями

Если при авторизации пользователь введёт почту, которая ещё не зарегистрирована, пользователь не сможет авторизироваться и появится сообщение об ошибке. На рисунке 4.6 показана форма авторизации, если введённой почты не существует.

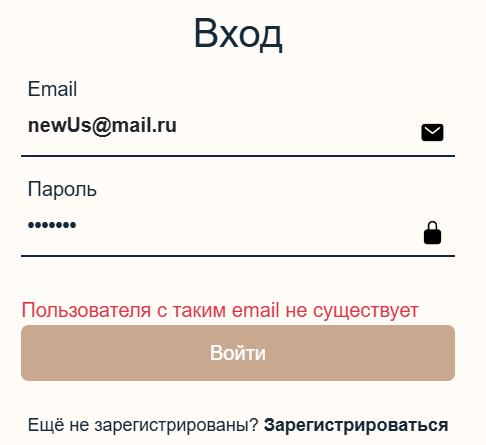


Рисунок 4.6 – Форма авторизации, если введённой почты не существует

Если при авторизации пользователь введёт неправильный пароль, пользователь не сможет авторизироваться и появится сообщение об ошибке. На рисунке 4.7 показана форма авторизации, если пользователь введёт неправильный пароль.

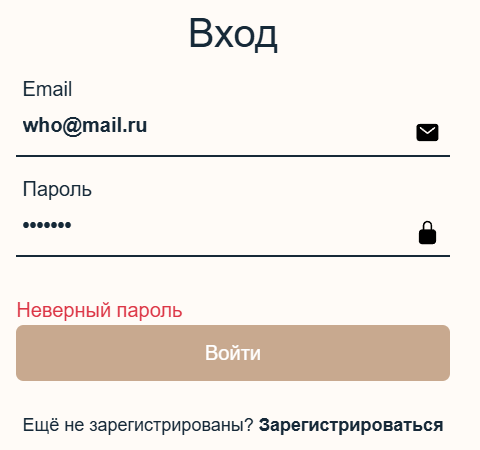


Рисунок 4.7 – Форма авторизации, если пользователь введёт неправильный пароль

При успешной регистрации или авторизации пользователь переходит на главную страницу, теперь ему доступны кнопки «Заказать», уведомления, личный кабинет, и его заказы.

Зарегистрированный пользователь не может сделать заказ, если он не заполнил поля ФИО и адреса в личном кабинете. На рисунке 4.8 показана вывод ошибки при оформлении заказа, если пользователь не заполнил данные в своём кабинете.

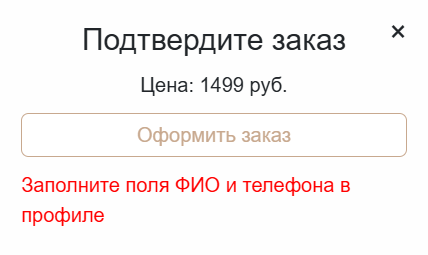


Рисунок 4.8 – Вывод ошибки при оформлении заказа, если пользователь не заполнил данные в своём кабинете

В своём кабинете пользователь не может обновить профиль, если он не введёт текущий пароль. На рисунке 4.9 представлен вывод предупреждения, если текущий пароль не введён.

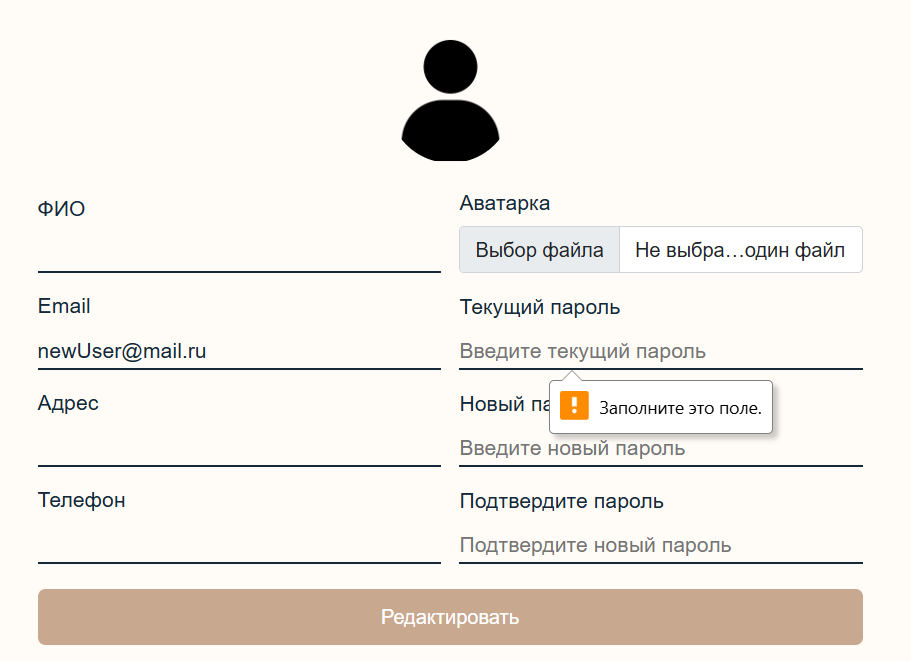


Рисунок 4.9 – Предупреждение, если текущий пароль не введён

Если пользователь введёт неправильный текущий пароль, то при нажатии на кнопку «Редактировать», ему выведется сообщение об ошибке. На рисунке 4.10 показано сообщение об ошибке, если текущий пароль неправильный.

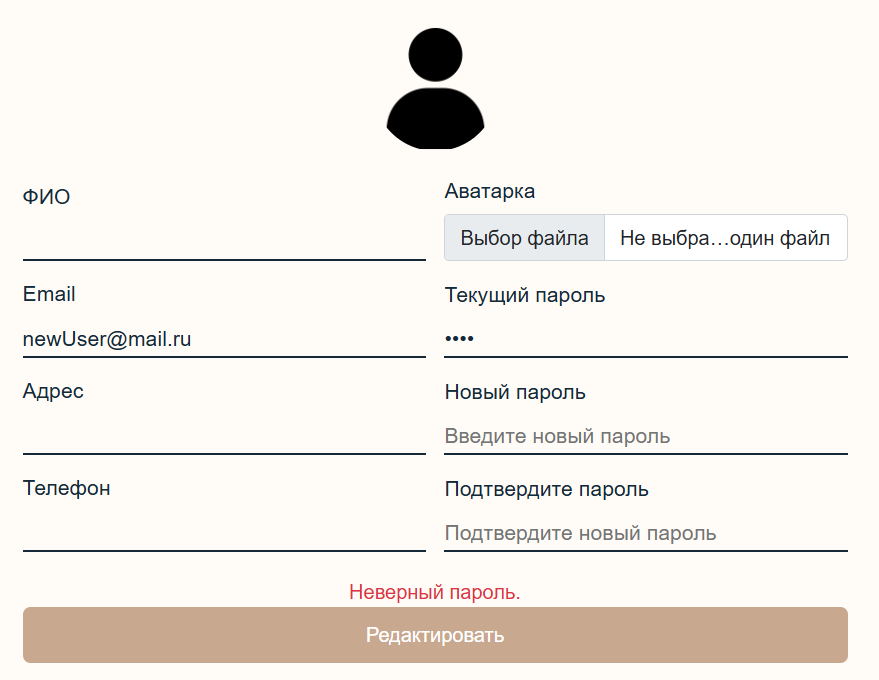


Рисунок 4.10 – Сообщение об ошибке, если текущий пароль неправильный

Также пользователь не может отредактировать профиль (и оформлять заказ), если поля ФИО и телефона не введены. На рисунке 4.11 представлено сообщение об ошибке, если ФИО и номер телефона не введены.

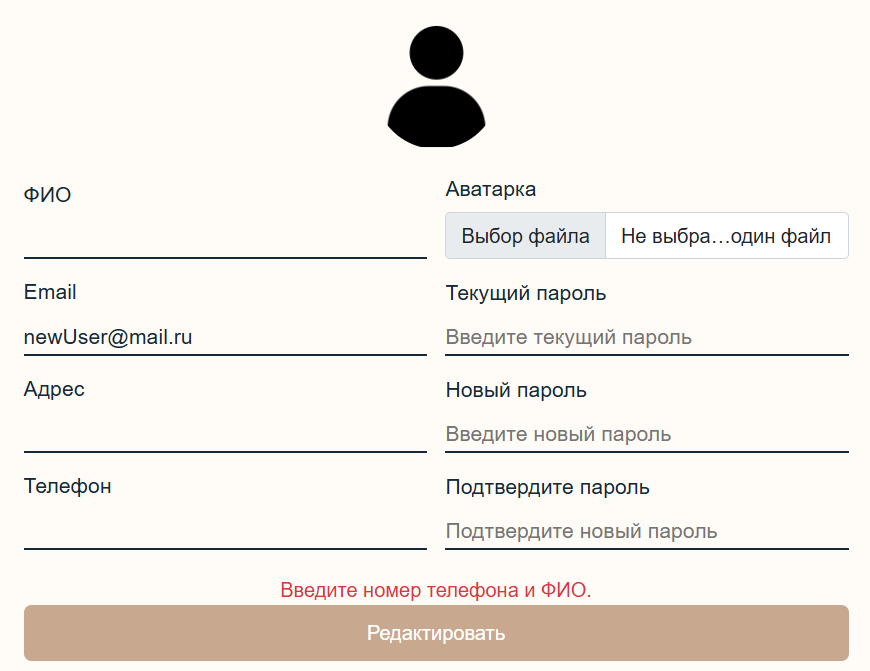


Рисунок 4.11 – Сообщение об ошибке, если ФИО и номер телефона не введены

Если пользователь неправильно ввёл номер телефона, появится сообщение об ошибке, которое представлено на рисунке 4.12.

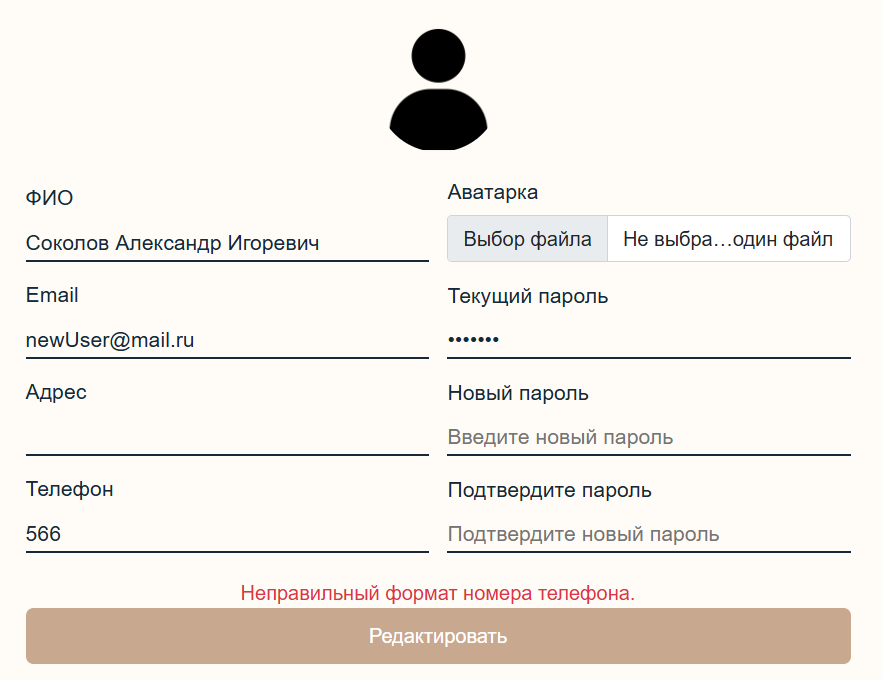


Рисунок 4.12 – Сообщение об ошибке, если номер телефона неправильного формата

Если пользователь хочет обновить пароль и неправильно повторяет пароль для его подтверждения появится сообщение об ошибке, которое представлено на рисунке 4.13.



Рисунок 4.13 – Сообщение об ошибке, если подтверждение пароля не совпадает

При правильном заполнении всех полей выводится сообщение об успешном обновлении данных, которое представлено на рисунке 4.14.

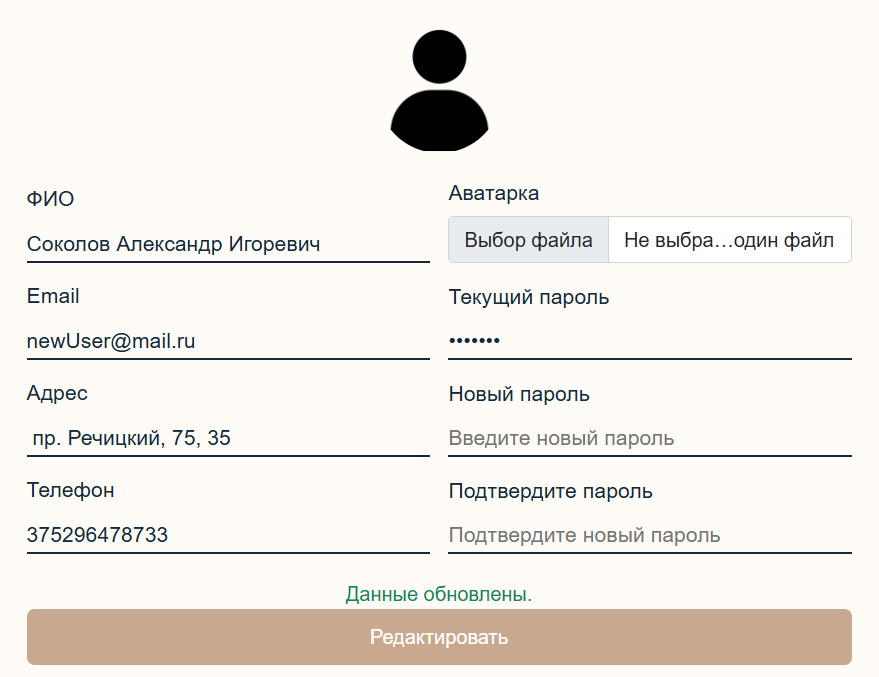


Рисунок 4.14 – Сообщение об ошибке об успешном обновлении данных

При успешном оформлении заказа, выводится текст, представленный на рисунке 4.15.

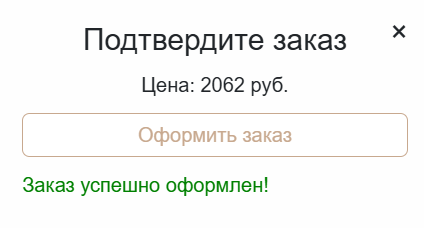


Рисунок 4.15 – Сообщение об успешном оформлении заказа

При успешном оформлении заказа с параметрами, выводится текст, представленный на рисунке 4.16.

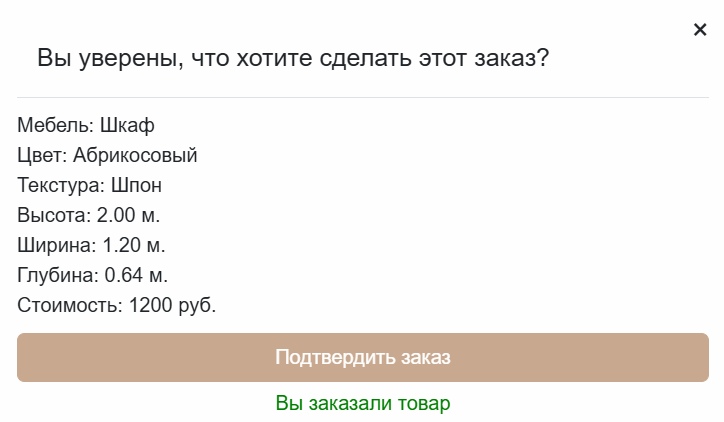


Рисунок 4.16 – Сообщение об успешном оформлении заказа с параметрами

При успешном добавлении нового товара на склад выводится текст, представленный на рисунке 4.17.



Рисунок 4.17 – Модальное окно с добавлением товара после успешного добавления нового товара на склад

При успешном добавлении существующего товара с другими параметрами выводится сообщение об успешном добавлении товара на склад. На рисунке 4.18 представлено модальное окно с сообщением после успешного добавления существующего товара с другим цветом.

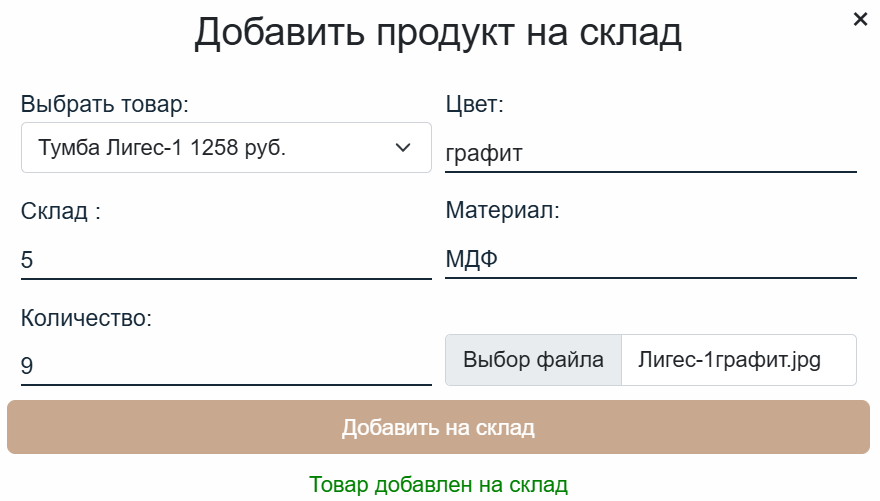


Рисунок 4.18 – Модальное окно после успешного добавления существующего товара с другим цветом

Склад после добавления нового товара представлен на рисунке 4.19.



Рисунок 4.19 – Страница склада после добавления товаров

После успешного редактирования товара выводится текст, представленный на рисунке 4.20.



Рисунок 4.20 – Модальное окно редактирования после успешного редактирования товара

После успешного удаления товара, в модальное окно удаления товара выводится текст, представленный на рисунке 4.21.

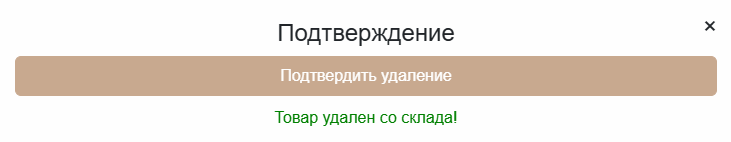


Рисунок 4.21 – Модальное окно удаления после успешного удаления товара

Список товаров на складе после удаления тумбы кремового цвета представлен на рисунке 4.22.



Рисунок 4.22 – Склад после удаления тумбы кремового цвета

# **ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОЙ РАБОТЫ**

## **Расчёт общей трудоёмкости разработки программного обеспечения**

Общий объём трудоёмкости разработки системы определяется исходя из количества и объёма функций, реализуемых программой, по каталогу функций ПО в соответствии с таблицей 1.1 приложения 1 источника [13].

Таблица 5.1 – Каталог функций программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| Код функции | Наименование (Содержание) функций | Объём функций (строк исходного кода (LOC)) | |
| По каталогу | Уточнённый |
| 101 | Организация ввода информации | 130 | 60 |
| 102 | Контроль, предварительная обработка и ввод информации | 490 | 135 |
| 107 | Организация ввода-вывода в интерактивном режиме | 280 | 200 |
| 109 | Управление вводом-выводом | 1970 | 330 |
| 201 | Генерация структуры базы данных | 3500 | 750 |
| 202 | Формирование базы данных | 1980 | 550 |
| 206 | Манипулирование данными | 7680 | 590 |
| 207 | Организация поиска и поиск в базе данных | 4720 | 200 |
| 506 | Обработка ошибочных сбойных ситуаций | 1540 | 240 |
| 507 | Обеспечение интерфейса между компонентами | 1680 | 350 |
| 703 | Расчет показателей | 420 | 170 |
| 801 | Простой поиск контента портала | 55 | 45 |
| 802 | Многокритериальный поиск контента портала | 85 | 60 |

Таким образом общий объём трудоёмкости разработки системы определяется по формуле (5.1):

где – объём отдельной функции ПО;

– общее число функций.

Анализируя разработанную программу, уточнённый объём ПО определяем по формуле (5.2):

где – уточнённый объём отдельной функции ПО в строках исходного кода (*LOC*).

Разработанное в ходе выполнения дипломной работы приложение относится к третьей категории сложности. На основании принятого к расчёту уточнённого объёма и категории сложности ПО определяется нормативная трудоемкость ПО выполняемых работ, которая приведена в таблице 5.2.

Таблица 5.2 – Нормативная трудоемкость на разработку ПО (

|  |  |  |
| --- | --- | --- |
| Уточнённый объём, | 3-я категория сложности ПО | Номер нормы |
| 3675 | 162 | 43 |

Дополнительные затраты труда, связанные с повышением сложности разрабатываемого ПО, учитываются посредством коэффициента повышения сложности ПО рассчитывается по формуле (5.3):

где – коэффициент, соответствующий степени повышения сложности;

𝑛 – количество учитываемых характеристик.

Таким образом:

Новизна разработанного ПО определяется путём экспертной оценки данных, полученных при сравнении характеристик разрабатываемого ПО с имеющимися аналогами. Влияние фактора новизны на трудоёмкость учитывается путём умножения нормативной трудоёмкости на соответствующий коэффициент, учитывающий новизну ПО . Разработанная программа обладает категорией новизны В, а значение = 0,63.

Современные технологии разработки компьютерных программ предусматривают широкое использование коробочных продуктов (пакетов, модулей, библиотек). Степень использования в разрабатываемом ПО стандартных модулей определяется их удельным весом в общем объеме ПО. В данном программном комплексе используется до 60% стандартных модулей, что соответствует значению коэффициента = 0,65.

Приложение разработано на языке *C#* в операционной системе *Windows*, что соответствует коэффициенту, учитывающему средства разработки ПО, = 1,0. Значения коэффициентов удельных весов трудоёмкости стадий разработки ПО в общей трудоемкости ПО определяются с учётом установленной категории новизны ПО. При этом сумма значений коэффициентов удельных весов всех стадий в общей трудоёмкости равна единице. Значения коэффициентов приведены в таблице 5.3.

Таблица 5.3 – Значения коэффициентов удельных весов трудоёмкости стадий разработки ПО в общей трудоёмкости ПО

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Категория новизны ПО | Без применения CASE-технологии | | | | |
| Стадии разработки ПО | | | | |
| ТЗ | ЭП | ТП | РП | ВН |
| Значения коэффициентов | | | | |
|  |  |  |  |  |
| В | 0,08 | 0,19 | 0,28 | 0,34 | 0,11 |

Нормативная трудоёмкость ПО выполняемых работ по стадиям разработки корректируется с учетом коэффициентов: повышения сложности ПО , учитывающих новизну ПО , учитывающих степень использования стандартных модулей , средства разработки ПО и определяются по формулам:

для стадии ТЗ по формуле (5.4):

для стадии ЭП по формуле (5.5):

для стадии ТП по формуле (5.6):

для стадии РП по формуле (5.7):

для стадии ВН по формуле (5.8):

где , , , и – значения коэффициентов удельных весов трудоёмкости стадий разработки ПО в общей трудоёмкости ПО.

Коэффициенты , , вводятся на всех стадиях разработки, а коэффициент вводится только на стадии РП.

Таким образом:

Общая трудоёмкость разработки ПО определяется суммированием нормативной (скорректированной) трудоёмкости ПО по стадиям разработки используя формулу (5.9):

где – нормативная (скорректированная) трудоёмкость разработки ПО на *i*-й стадии (чел.-дн.);

𝑛 – количество стадий разработки.

Таким образом:

Результаты расчётов по определению нормативной и скорректированной трудоёмкости ПО по стадиям разработки и общую трудоёмкость разработки ПО представлены в таблице 5.4.

Таблица 5.4 – Расчёт общей трудоёмкости разработки программного обеспечения

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Показатели | Стадии разработки | | | | | Итого |
| ТЗ | ЭП | ТП | РП | ВН |
| Общий объём ПО , кол-во строк LOC | − | − | − | − | − | 24530 |
| Общий уточнённый объем ПО , кол-во строк LOC | − | − | − | − | − | 3675 |
| Категория сложности разрабатываемого ПО | − | − | − | − | − | 3 |
| Нормативная трудоёмкость разработки ПО , чел.-дн. | − | − | − | − | − | 162 |
| Коэффициент повышения сложности ПО | 1,25 | 1,25 | 1,25 | 1,25 | 1,25 | − |
| Коэффициент, учитывающий новизну ПО | 0,63 | 0,63 | 0,63 | 0,63 | 0,63 | − |
| Коэффициент, учитывающий степень использования стандартных модулей | − | − | − | 0,65 | − | − |
| Коэффициент, учитывающий средства разработки ПО | 1,0 | 1,0 | 1,0 | 1,0 | 1,0 | − |
| Коэффициенты удельных весов трудоёмкости стадий разработки ПО (, , , , ) | 0,08 | 0,19 | 0,28 | 0,34 | 0,11 | 1,0 |
| Распределение нормативной трудоёмкости ПО по стадиям, чел.-дн. | 13 | 31 | 45 | 55 | 18 | 162 |
| Распределение скорректированной (с учётом , , , ) трудоёмкости ПО по стадиям, чел.-дн. | 10 | 24 | 35 | 28 | 14 | − |
| Общая трудоемкость разработки ПО , чел.-дн. | − | − | − | − | − | 111 |

## **Расчёт затрат на разработку программного продукта**

В состав затрат на разработку системы по автоматизации учёта лабораторных испытаний промышленного предприятия входят следующие статьи расходов:

* затраты труда на создание программного продукта (затраты по основной, дополнительной заработной плате и соответствующие отчисления),;
* затраты на изготовление эталонного экземпляра ;
* затраты на технологию (затраты на приобретение и освоение программных средств, используемых при разработке программного продукта; затраты на ПО, используемое как эталон) ;
* затраты на машинное время (расходы на содержание и эксплуатацию технических средств разработки, эксплуатации и сопровождения) ;
* затраты на материалы (информационные носители) ;
* затраты на энергию, на использование каналов связи (для отдельных видов);
* общепроизводственные расходы (затраты на управленческий персонал, на содержание помещений) ;
* непроизводственные (коммерческие) расходы (затраты, связанные с рекламой, поиском заказчиков, поставками конкретных экземпляров) .

В стоимость оборудования ; входят расходы на его приобретение по прейскурантам, прайс-листам и другим источникам, а также расходы на приемку и хранение оборудования (примерно 2 % от стоимости). Также в стоимость оборудования включаются транспортно-заготовительные расходы, т. е. расходы по его доставке и стоимость монтажа устанавливаемого оборудования. Как правило, их принимают в размере 5-10 % от стоимости нового оборудования.

Стоимость приобретения ПЭВМ на 2024 год равна 3299 руб. согласно [14]. Таким образом получаем стоимость оборудования:

Суммарные затраты на разработку ПО определяются по формуле (5.10):

Расходы на оплату труда разработчиков с отчислениями определяются по формуле (5.11):

где – основная заработная плата разработчиков, руб.;

– дополнительная заработная плата разработчиков, руб.;

– сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.

Основная заработная плата разработчиков рассчитывается по формуле (5.12):

где – средняя часовая тарифная ставка;

– общая трудоемкость разработки, чел.-ч;

– коэффициент, учитывающий доплаты стимулирующего характера.

Средняя часовая тарифная ставка определяется по формуле (5.13):

где – часовая тарифная ставка разработчика *i* – й категории, руб./ч;

– количество разработчиков i-й категории.

Часовая тарифная ставка разработчика *i*-й категории вычисляется путём деления месячной тарифной ставки на установленный при восьмичасовом рабочем дне по формуле (5.14):

где – фонд рабочего времени 168ч;

– месячная тарифная ставка;

– часовая тарифная ставка ставка разработчика *i* – й категории, руб./ч.

Базовая ставка в РБ на 2024 год составляет 250 руб., согласно [15], а тарифный коэффициент равен 1,57. Таким образом:

где 2,43 – коэффициент приведения базовой тарифной ставки первого разряда в сфере ИТ;

Находим :

где коэффициент 8 – количество рабочих часов в день;

1,5 – коэффициент премий.

Дополнительная заработная плата определяется по формуле (5.15):

где – норматив отчислений на дополнительную заработную плату разработчиков.

Таким образом:

Отчисления от основной и дополнительной заработной платы (отчисления на социальные нужды и обязательное страхование) рассчитываются по формуле (5.16):

где – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы (= 34,6 %).

Затраты машинного времени определяются по формуле (5.17):

где – стоимость 1 часа машинного времени (руб./ч.);

– коэффициент мультипрограммности, показывающий распределение времени работы ЭВМ в зависимости от количества пользователей ЭВМ; =1;

– машинное время ЭВМ, необходимое для разработки и отладки проекта (ч.)

где – затраты на заработную плату обслуживающего персонала с учётом всех отчислений, (руб. в год);

– стоимость аренды помещения под размещение вычислительной техники, (руб. в год);

– амортизационные отчисления за год, (руб. в год);

– затраты на электроэнергию, (руб. в год);

– затраты на материалы, необходимые для обеспечения нормальной работы ПЭВМ (вспомогательные), (руб. в год);

– затраты на текущий и профилактический ремонт ЭВМ (руб. в год);

– прочие затраты, связанные с эксплуатацией ПЭВМ (руб. в год);

– действительный фонд времени работы ЭВМ (час/год).

Затраты на заработную плату обслуживающего персонала определяются с использованием формул (5.19) – (5.22):

где – основная заработная плата обслуживающего персонала, руб.;

– дополнительная заработная плата обслуживающего персонала, руб.;

– сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.;

– количество обслуживаемых ПЭВМ, шт.;

– месячная тарифная ставка *i*-го работника, руб.;

𝑛 – численность обслуживающего персонала, чел.;

– процент дополнительной заработной платы обслуживающего персонала от основной;

– процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы.

Годовые затраты на аренду помещения определяются по формуле (5.23):

где – средняя годовая ставка арендных платежей, руб./;

𝑆 – площадь помещения, ;

– количество ПЭВМ, шт.

Следовательно, получаем годовые затраты на аренду помещения:

где 17,76 – базовая арендная величина.

Сумма годовых амортизационных отчислений (ЗАМ) определяется по формуле (5.24):

где – затраты на приобретение (стоимость) единицы ПЭВМ, руб.;

– коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, = 12 − 13% от ;

∙(1 + ) – балансовая стоимость ЭВМ, руб.;

*m* – количество оборудования;

– норма амортизации для вида ЭВМ;

– норма амортизации, %.

Стоимость электроэнергии, потребляемой за год, определяется по формуле (5.25):

где – паспортная мощность ПЭВМ, (кВт);

– действительный годовой фонд времени работы ПЭВМ, = 1714,6 ч., согласно производственному календарю на 2024 год;

– стоимость одного кВт-часа электроэнергии, руб;

*A* – коэффициент интенсивного использования мощности, *А=*0.98...0.9

Затраты на материалы , необходимые для обеспечения нормальной работы ПЭВМ составляют около 1% от балансовой стоимости ЭВМ и определяются формулой (5.26):

где – затраты на приобретение (стоимость) ЭВМ, руб.;

– коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования,= 12 − 13% от ;

– коэффициент, характеризующий затраты на вспомогательные материалы (= 0,01).

Затраты на материалы (носители информации, распечатка и прочее), необходимые для обеспечения нормальной работы ПЭВМ рассчитываются по формуле (5.27):

где – цена *i*-го наименования материала полуфабриката, комплектующего, руб.;

– потребность в *i*-м материале, полуфабрикате, комплектующем, натур. ед.;

– коэффициент, учитывающий сложившийся процент транспортно-заготовительных расходов в зависимости от способа доставки товаров, ;

– цена возвратных отходов *i*-го наименования материала, руб.;

– количество возвратных отходов *i*-го наименования, натур. ед.;

*n* – количество наименований материалов, полуфабрикатов, комплектующих.

Таким образом:

где 42 – стоимость печати, руб.

Затраты на текущий и профилактический ремонт (ЗТР) принимаются равными 5%–9% от балансовой стоимости ЭВМ и рассчитываются по формуле (5.28):

где – коэффициент, характеризующий затраты на текущий и профилактический ремонт ( = 0,08)

Прочие затраты, связанные с эксплуатацией ЭВМ , состоят из амортизационных отчислений на здания, стоимости услуг сторонних организаций, составляют 5 % от балансовой стоимости и рассчитываются по формуле (5.29):

где – коэффициент, характеризующий размер прочих затрат, связанных с эксплуатацией ЭВМ ( = 0,05).

Для расчета машинного времени ЭВМ (в часах), необходимого для разработки и отладки проекта, следует использовать формулу (5.30):

где – срок реализации стадии «Рабочий проект» (РП), 28 дней;

– срок реализации стадии «Ввод в действие» (ВН), 14 дня;

– продолжительность рабочей смены, (ч.), = 8 ч.;

– количество рабочих смен, = 1.

При написании дипломной работы были использованы среда разработки Visual Studio 2022, причём для нужд разработки и тестирования хватило бесплатного тарифа, поэтому затраты на технологию и изготовление эталонного экземпляра будут нулевыми.

Общепроизводственные затраты рассчитываются по формуле (5.31):

где – норматив общепроизводственных затрат равный 5%.

Непроизводственные затраты рассчитываются по формуле (5.32):

где – норматив непроизводственных затрат равный 3%.

Итого получаем суммарные затраты на разработку:

Все статьи затрат на разработку программного обеспечения отражены в таблице 5.5.

Таблица 5.5 – Таблица статей затрат на разработку программного обеспечения

|  |  |
| --- | --- |
| Статья затрат | Итого |
| 1 | 2 |
| Затраты на оплату труда разработчиков , руб. |  |
| Основная заработная плата разработчиков |  |
| Дополнительная заработная плата разработчиков |  |
| Отчисления от основной и дополнительной ЗП |  |
| Затраты машинного времени |  |
| Стоимость машино-часа |  |
| Затраты на заработную плату обслуживающего персонала |  |
| Годовые затраты на аренду помещения |  |
| Сумма годовых амортизационных отчислений |  |
| Стоимость электроэнергии, потребляемой за год |  |
| Действительный годовой фонд времени работы ПЭВМ | 1713,6 |
| Затраты на материалы |  |
| Затраты на текущий и профилактический ремонт |  |
| Прочие затраты, связанные с эксплуатацией ЭВМ |  |

Продолжение таблицы 5.5

|  |  |
| --- | --- |
| 1 | 2 |
| Машинное время ЭВМ |  |
| Затраты на изготовление эталонного экземпляра | 0 |
| Затраты на технологию | 0 |
| Затраты на материалы | 42 |
| Общепроизводственные затраты |  |
| Непроизводственные (коммерческие) затраты |  |
| Суммарные затраты на разработку ПО |  |

## **Формирование цены при создании программного обеспечения**

Оптовая цена ПО , определяется следующей формулой (5.33):

где – себестоимость ПО, руб.;

– прибыль от реализации ПО, руб.;

– уровень рентабельности ПО, % ( = 30 %)

Прогнозируемая отпускная цена ПО с НДС рассчитывается по формуле (5.34):

Налог на добавленную стоимость рассчитывается по формуле (5.35):

где – ставка налога на добавленную стоимость, %, = 20 %.

Розничную цену на программный продукт можно определить по формуле (5.36):

где – торговая наценка при реализации программного обеспечения через специализированные магазины (торговых посредников), ее значение принимается равным 10%.

Результаты расчетов формирования цены на разработку программы приведены в таблице 5.6. Эти значения позволяют оценить рентабельность приложения в сравнении с аналогами, при наличии подходящего аналога и данных о его стоимости. Для этой оценки будет использовано значение розничной цены. Таблица включает итоги расчёта стоимости различных статей расходов, по формулам приведённым выше.

Таблица 5.6 – Расчет формирования цены на разработку программы

|  |  |
| --- | --- |
| Наименование статьи расходов | Значение |
| Полная себестоимость |  |
| Прибыль от реализации ПО |  |
| Отпускная цена ПО без НДС |  |
| Налог на добавленную стоимость |  |
| Отпускная цена ПО с НДС |  |
| Розничная цена |  |

## **Расчёт эффекта от внедрения программного продукта**

Для того, чтобы рассчитать годовой экономический эффект от использования нового ПО необходимо такие параметры как заработная плата специалиста, работающего с программой, стоимость 1 часа работы этого специалиста и время, сэкономленное при использовании программы.

Эффект (прибыль) может просчитываться по формуле (5.37):

где – текущие и инвестиционные затраты по базовому варианту, включающие затраты на приобретение продукта (цену), его эксплуатацию;

З – текущие и инвестиционные затраты.

По результатам изучения рыночных цен программных продуктов схожего функционального назначения было установлено, что средняя стоимость аналога составляет 45000 руб. Таким образом, эффект:

На основе рассчитанного эффекта от разработки программного обеспечения следует рассчитать следующие итоговые показатели, характеризующие экономическую эффективность проекта, которые можно рассчитать по формуле (5.38), рентабельность затрат (З) или инвестиций (И) на новую информационную технологию, программный продукт:

Таким образом, рентабельность:

Срок окупаемости затрат (инвестиций) рассчитывается по формуле (5.39):

Таким образом, срок окупаемости затрат:

Т.к. срок окупаемости составляет меньше одного календарного года, то проведение динамической оценки (расчёт динамических показателей эффективности) не целесообразно.

Годовой экономический эффект определяется по формуле (5.40):

где – рентабельность затрат (инвестиций) базового варианта, 30%.

Таким образом, годовой экономический эффект:

На основании выполненных расчетов была сформирована таблица 5.7 технико-экономических показателей проекта. После оценки технико-экономических показателей проектного программного обеспечения можно сделать вывод о том, что реализация проекта является обоснованной и экономически целесообразной, так как срок окупаемости проекта меньше года при размере годового экономического эффекта 12404,28 руб. с уровнем рентабельности 106,72%.

Таблица 5.7 – Технико-экономические показатели проекта

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование параметра | Условные обозначения | Базовый вариант | Новый вариант |
| Исходные данные | | | |
| Общая трудоемкость разработки ПО | чел.-дн. | − | 111 |
| Затраты на разработку ПО | руб. | − | 16168,52 |
| Затраты на оплату труда разработчиков | руб. | − |  |
| Затраты машинного времени | руб. | − |  |
| Затраты на материалы | руб. | − | 42 |
| Общепроизводственные затраты | руб. | − |  |
| Непроизводственные затраты | руб. | − |  |
| Отпускная цена ПП с НДС | руб. | − |  |
| Розничная цена ПП | руб. | − |  |
| Рентабельность затрат | % | − | 106,72% |
| Простой срок окупаемости проекта | лет | − | 0,94 |
| Годовой экономический эффект | руб. | − | 12404,28 |

# **ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ**

## **Государственное управление охраной труда**

Государственное управление охраной труда – деятельность субъектов государственного управления охраной труда, направленная на реализацию конституционных прав граждан на здоровые и безопасные условия труда.

Согласно статье 1 Закона «Об охране труда» Республики Беларусь [16], охрана труда – система обеспечения жизни и здоровья работников в процессе трудовой деятельности, включающая правовые, социально-экономические, организационные, технические, психофизиологические, санитарно-противоэпидемиологические, лечебнопрофилактические, реабилитационные и иные мероприятия и средства.

Основным принципом государственной политики в нашей стране в области охраны труда является приоритет жизни и здоровья работников по отношению к результатам трудовой деятельности, установление ответственности нанимателей за безопасность труда, совершенствование правовых отношений и механизмов в этой сфере.

Основным нормативным документом в области охраны труда является Закон Республики Беларусь «Об охране труда», который был принят в 2008 году. Согласно [17], на основании данного закона создана целостная и взаимоувязанная система управления охраной труда, определяющая роль и задачи каждого из участников процесса обеспечения безопасных условий труда на всех уровнях: от республиканского уровня до конкретных организаций, должностных лиц и работников.

Она позволяет комплексно решать вопросы обеспечения безопасности на рабочих местах, создавая необходимые условия для реализации права на работу в безопасных условиях труда, а также на защиту работником своих прав и получение соответствующих компенсаций при наступлении несчастного случая на производстве или профессионального заболевания.

Система обязательного страхования от несчастных случаев на производстве, формируемая за счет взносов работодателей, обеспечивает социальную защиту работников, получивших производственную травму или профессиональное заболевание.

В республике на законодательном уровне урегулированы вопросы обучения работников и руководителей по вопросам охраны труда, проведения медицинских осмотров работающих, предоставления средств индивидуальной и коллективной защиты, расследования несчастных случаев на производстве.

Осуществляется мониторинг вредных условий труда путем проведения в организациях аттестации рабочих мест по условиям труда. Работникам, работающим во вредных условиях труда, предоставляются компенсации по условиям труда: право на повышенную оплату труда, сокращенный рабочий день, дополнительный трудовой отпуск.

Для отдельных категорий работников предусмотрена возможность более раннего выхода на пенсию – на 5 и 10 лет. В этих целях с 2009 года в республике специальным законом введена система профессионального пенсионного страхования, основанная на дополнительных взносах работодателей в зависимости от наличия у них рабочих мест с вредными условиями труда.

В целях обеспечения прав граждан на безопасный труд в Республике Беларусь с 1993 года введен институт государственного надзора за соблюдением законодательства об охране труда.

В настоящее время его осуществляет специально созданный надзорный орган – Департамент государственной инспекции труда Министерства труда и социальной защиты Республики Беларусь.

В решении вопросов охраны труда активно участвуют и социальные партнеры. Закон Республики Беларусь «О профессиональных союзах» наделил профсоюзы правами по защите прав работников на безопасные условия труда в рамках общественного контроля.

Меры по обеспечению безопасного труда являются неотъемлемой частью общей системы управления организацией. В этой работе учитываются особенности конкретного предприятия, те риски, с которыми связаны осуществляемые на нем производственные процессы.

Учесть данные особенности и обеспечить комплексную работу по управлению рисками в области охраны труда призвана система управления охраной труда. Закон Республики Беларусь «Об охране труда» обязывает нанимателя разрабатывать, внедрять и поддерживать функционирование системы управления охраной труда.

Для оказания помощи нанимателям разработаны Рекомендациями по разработке системы управления охраной труда в организации, действует национальный стандарт СТБ *ISO* 45001-2020 «Системы менеджмента здоровья и безопасности при профессиональной деятельности. Требования и руководство по применению» [18].

Весь комплекс мероприятий в области охраны труда, от принятия законодательства и разработки систем управления охраной труда до государственного надзора и общественного контроля, направлен на решение одной главной задачи – сокращение производственного травматизма. В Республике Беларусь в этом вопросе удалось достичь положительных результатов.

## **Цели и задачи государственного управления охраны труда**

Целью государственного управления охраной труда является создание условий, обеспечивающих сохранение жизни и здоровья граждан в процессе трудовой деятельности.

Главная цель государственного управления охраной труда состоит в том, чтобы предотвращать производственные несчастные случаи и заболевания, связанные с работой. Для этого разрабатываются нормативные акты, проводятся инспекции и контроль за соблюдением требований безопасности и охраны труда.

Для достижения цели государственного управления охраной труда на современном этапе необходимо решение следующих задач:

* уточнение функций и ответственности субъектов государственного управления охраной труда;
* усиление профилактической направленности решений субъектов государственного управления охраной труда на всех уровнях;
* совершенствование государственного управления охраной труда, внедрение экономических механизмов и методов прогнозирования в этой сфере;
* сертификация систем управления охраной труда в организациях;
* включение системы управления охраной труда в общую систему управления производством;
* совершенствование нормативного правового обеспечения охраны труда.

Направления государственной политики – это основные цели и задачи, которые государство ставит перед собой в определенной области в рамках своей деятельности. Они определяются на основе потребностей и приоритетов общества и могут быть различными в различных областях.

Согласно [19], основные направления государственной политики в области охраны труда:

* государственное управление деятельностью в области охраны труда, включая государственный надзор и контроль за соблюдением законодательства по этим вопросам;
* принятие законов и иных нормативных правовых актов, направленных на совершенствование правоотношений в этой сфере, установление единых нормативных требований в области безопасности и гигиены труда;
* комплексное решение задач обеспечения здоровых и безопасных условий труда с учетом других направлений социальной и экономической политики, достижений в области науки, техники, технологий и охраны окружающей среды;
* организация научно-исследовательских работ по вопросам безопасности и гигиены труда, участие государства в финансировании республиканских программ по улучшению условий и охраны труда;
* использование экономического механизма в управлении охраной труда, проведение налоговой политики, стимулирующей создание здоровых и безопасных условий труда, разработку и внедрение безопасных техники и технологий, эффективных средств защиты;
* взаимодействие и сотрудничество органов государственного управления с нанимателями и профессиональными союзами в разработке и реализации государственной политики в области охраны труда, содействие общественному контролю за соблюдением прав и интересов работников в области охраны труда;
* обеспечение социально-экономической защиты прав работников в области охраны труда, в том числе потерпевших на производстве, а также членов их семей на основе обязательного социального страхования от несчастных случаев на производстве, профессиональных заболеваний, установление компенсаций за тяжелую работу и работу с вредными и (или) опасными условиями труда;
* подготовка специалистов, обучение и повышение квалификации работников по вопросам охраны труда;
* организация государственной статистической отчетности, создание системы информации и мониторинга о состоянии условий и охраны труда;
* международное сотрудничество, создание условий для ратификации конвенций Международной организации труда, других международных организаций, межгосударственных договоров и соглашений в области охраны труда.

## **Органы государственного управления охраной труда и их полномочия**

Государственное управление в области охраны труда осуществляют Президент Республики Беларусь, Правительство Республики Беларусь, республиканские органы государственного управления и иные государственные организации, подчиненные Правительству Республики Беларусь, местные исполнительные и распорядительные органы в пределах своей компетенции.

Президент Республики Беларусь определяет единую государственную политику в области охраны труда и осуществляет иные 35 полномочия в этой области в соответствии с Конституцией Республики Беларусь, Законом «Об охране труда» и иными законодательными актами.

Правительство Республики Беларусь обеспечивает проведение единой государственной политики в области охраны труда, в пределах своей компетенции определяет полномочия республиканских органов государственного управления и иных государственных организаций, подчиненных Правительству Республики Беларусь, в области охраны труда, осуществляет иные полномочия в этой области в соответствии с Конституцией Республики Беларусь, настоящим Законом, иными законами и актами Президента Республики Беларусь.

Республиканские органы государственного управления и иные государственные организации, подчиненные Правительству Республики Беларусь, осуществляют управление деятельностью подчиненных (входящих в состав (систему)) организаций по вопросам охраны труда посредством реализации полномочий собственника с анализом эффективности работы подчиненных организаций в области охраны труда и выработкой предложений по ее повышению.

Особое место занимает Министерство труда и социальной защиты Республики Беларусь. Основные задачи:

* разработка предложений и реализация основных направлений и приоритетов государственной политики по вопросам содействия занятости населения, оплаты, условий и охраны труда, государственного социального страхования и пенсионного обеспечения, демографической безопасности, социального партнерства, урегулирования коллективных трудовых споров, социального обслуживания и социальной помощи, альтернативной службы, улучшения социально-экономических условий жизнедеятельности семьи, обеспечения равных прав и равных возможностей мужчин и женщин, социальных гарантий населению;
* совершенствование законодательства о труде, занятости и социальной защите населения;
* осуществление контроля за соблюдением законодательства о государственном социальном страховании, надзора за соблюдением законодательства о труде и об охране труда;
* осуществление международного сотрудничества в вышеуказанных областях.

Советы, исполнительные и распорядительные органы в пределах своей компетенции в порядке, установленном законодательством, принимают меры по обеспечению социальной защиты граждан на соответствующей территории, в том числе путем предоставления государственной адресной социальной помощи, назначения и выплаты пенсий, пособий, оказания иных видов социальной поддержки.

## **Инструкции по охране труда**

Важнейшим организационным мероприятием по профилактике производственного травматизма и профессиональных заболеваний, обеспечению конституционного права граждан на здоровые и безопасные условия труда являются обучение, инструктаж и проверка знаний по охране труда работников. Организацию обучения, инструктажа и проверки знаний по вопросам охраны труда работников определяет «Инструкция о порядке подготовки (обучения), переподготовки, стажировки, инструктажа, повышения квалификации и проверки знаний работающих по вопросам охраны труда», утвержденная постановлением Министерства труда и социальной защиты Республики Беларусь от 28 ноября 2008 г. № 175 (далее – Инструкция). Данная Инструкция устанавливает порядок и виды обучения, инструктажей и проверки знаний по вопросам охраны труда руководителей и специалистов, рабочих, учащихся, студентов, воспитанников, распространяется на всех работодателей (нанимателей) независимо от форм собственности и сферы хозяйственной деятельности и является обязательной для исполнения всеми хозяйствами, предприятиями, учреждениями и организациями Республики Беларусь.

В соответствии со статьей 226 Трудового кодекса РБ [20] наниматель обязан обеспечивать подготовку (обучение), инструктаж, повышение квалификации и проверку знаний работников по вопросам охраны труда.

Перед проверкой знаний по вопросам охраны труда организуются курсы, семинары, лекции, собеседования, консультации другие занятия.

В организации приказом ее руководителя создается комиссия для проверки знаний по вопросам охраны труда. В ее состав включаются специалисты службы охраны труда, юридической службы, главный энергетик, главный механик, главный технолог и другие главные специалисты, уполномоченные представители профсоюзов, представители государственных органов надзора и контроля (при необходимости и по согласованию с этими органами).

Руководитель организации, его заместители, ответственные за организацию охраны труда, главные специалисты, руководители (специалисты) служб охраны труда, члены комиссии организации, иные должностные лица организации проходят проверку знаний по вопросам охраны труда в комиссии вышестоящей организации или комиссии республиканского органа государственного управления, иной государственной организации, подчиненной Правительству Республики Беларусь. Лица, занимающиеся предпринимательской деятельностью (наниматели), проверку 13 знаний по вопросам охраны труда проходят в комиссиях местных исполнительных и распорядительных органов.

Лица, не прошедшие проверку знаний по вопросам охраны труда, проходят повторную проверку в срок не более одного месяца со дня ее проведения. Лица, не прошедшие проверку знаний по вопросам охраны труда повторно, не допускаются к выполнению работ.

Рабочие, занятые на работах с повышенной опасностью, проходят периодическую проверку знаний по вопросам охраны труда не реже одного раза в год.

Не позднее месяца со дня назначения на должность и периодически не реже одного раза в три года руководители и специалисты проходят проверку знаний по вопросам охраны труда в соответствующих комиссиях. Перечень должностей руководителей и специалистов, которые должны проходить проверку знаний по вопросам охраны труда, утверждает руководитель организации. Проверка знаний по вопросам охраны труда данной категории работающих проводится с учетом их должностных обязанностей и характера производственной деятельности. Лицам, прошедшим проверку знаний, выдается удостоверение по охране труда.

Инструкции по охране труда являются нормативными документами, устанавливающими требования безопасности при выполнении работниками работ в производственных помещениях, на территории предприятия, на строительных площадках и в иных местах, где работники выполняют порученную им работу или служебные обязанности. Они разрабатываются для работников отдельных профессий и на отдельные виды работ, относящиеся к перечню работ с повышенной опасностью. Инструкции включают только те требования, которые касаются безопасности труда и выполняются самим работником. Инструкции разрабатываются на основе стандартов безопасности труда, правил и норм безопасности и гигиены труда, типовых инструкций, требований безопасности, изложенных в эксплуатационной и ремонтной документации заводов-изготовителей оборудования, используемого на предприятии, а также на основе технологической документации предприятия с учетом конкретных условий производства.

Требования инструкции излагаются в соответствии с последовательностью технологического процесса и с учётом условий, в которых выполняется данная работа. Инструкция должна содержать следующие разделы:

* общие требования безопасности;
* требования безопасности перед началом работы;
* требования безопасности при выполнении работы;
* требования безопасности в аварийных ситуациях;
* требования безопасности по окончании работы.

# **РЕСУРСО- И ЭНЕРГОСБЕРЕЖЕНИЕ ПРИ ВНЕДРЕНИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **Основные понятия в области ресурсо- и энергосбережения**

Ресурсосбережение – это совокупность мер по бережливому и эффективному использованию факторов производства (капитала, земли, труда). Обеспечивается посредством использования ресурсосберегающих и энергосберегающих технологий. Например, снижения фондоёмкости и материалоемкости продукции, повышения производительности труда, сокращения затрат живого и овеществленного труда, повышения качества продукции, рационального применения труда менеджеров и маркетологов, использования выгод международного разделения труда и др.

Ресурсы представляют собой ценности, запасы, возможности и источники дохода в государственном бюджете. Их можно разделить на природные и экономические, включая материальные, трудовые и финансовые ресурсы.

В настоящее время ресурсосбережение – одна из приоритетных задач экономики РБ. Это связано прежде всего с дефицитом многих видов ресурсов: ростом стоимости их добычи (подготовки), а также серьезными экологическими проблемами.

Необходимо отметить, что одним из факторов, обеспечивающих ресурсосбережение, являются инновации. Активная инновационная деятельность необходима предприятиям отрасли для обеспечения конкурентоспособности в долгосрочной перспективе. Но при этом необходимо учитывать, что отдельные виды инноваций по-разному влияют на ресурсосберегающую деятельность предприятия. Наибольшее влияние оказывают технологические и производственные инновации. В наименьшей степени оказывают влияние социальные, маркетинговые и организационные инновации.

Энергосбережение – это комплекс мер по реализации правовых, организационных, научных, производственных, технических и экономических мер, направленных на эффективное (рациональное) использование (и экономное расходование) топливно-энергетических ресурсов и на вовлечение в хозяйственный оборот возобновляемых источников энергии.

Основной целью энергоресурсосбережения является повышение энерго- и ресурсоэффективности всех отраслей, во всех пунктах населения, а также в стране в целом.

Энергетическая эффективность – это характеристики, отражающие отношение полезного эффекта от использования энергетических ресурсов к затратам энергетических ресурсов, произведенным в целях получения такого эффекта, применительно к продукции, технологическому процессу, юридическому лицу, индивидуальному предпринимателю.

Республиканским органом государственного управления, уполномоченным Правительством Республики Беларусь для проведения государственной политики в сфере энергосбережения, является Комитет по энергоэффективности при Совете Министров Республики Беларусь. Основными задачами Комитета по энергоэффективности при Совете Министров Республики Беларусь являются:

* проведение государственной политики в сфере ресурсо- и энергосбережения;
* осуществление государственного надзора за рациональным использованием ресурсов, топлива, электрической и тепловой энергии.

## **Ресурсо- и энергосбережения в результате внедрения разработанного ПО**

Внедрение ПО может позволить автоматизировать некоторые рабочие процессы, что уменьшает необходимость в ручной работе и, как следствие, сокращает время и затраты на выполнение задач.

При разработке ПО важно понимать, что любая возможность программного продукта стоит денег как для её разработки и внедрения, так и для последующего сопровождения. Чем больше функциональных возможностей закладывается в продукт, тем он будет дороже в создании и в дальнейшем сопровождении. Поэтому [при разработке сайтов, *web*‑приложений и сервисов](https://web-creator.ru/services/development) стоит задумываться о необходимости реализации тех или иных функций с коммерческой точки зрения.

Ещё один аспект «платы за широкие возможности» – это баги (ошибки и сбои), чем больше и сложнее продукт, чем активнее внедряются в него новые функции, тем больше вероятность, что в нём будут ошибки. Если привести аналогию, то «вычитать» литературный текст объёмом в один лист A4 можно достаточно быстро и ошибок там избежать просто, а вот с многостраничным документом уже часто возникают сложности. В разработке ПО всё усложняется тем, что компонентов в разрабатываемых продуктах достаточно много и они тесно взаимосвязаны – два вполне корректно работающих по отдельности компонента, например, могут при сочетании приводить к сбоям, а добавление чего‑либо в одном месте – ошибки в другом. В какой‑то мере этих проблем можно избежать при использовании автоматического тестирования иметодологии *TDD* (*test-driven development*). *TDD* техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки.

Как правило, подавляющее большинство «дополнительных» функций программных продуктов неэффективны с коммерческой точки зрения – затраты на них существенные, а вот финансовой выгоды они не приносят.

Добавление функций в программный продукт всегда вызывает дополнительные изменения в проекте, а эти изменения не всегда очевидны. Влияние расширения функционала на различные аспекты разработки зависит и от того, когда изменения поступили, и от того, каков их объём, и от их связанности с другими функциями программного продукта.

Зачастую приложения запрашивают с сервера большие списки с данными, которые необходимо отобразить. Здесь появляется несколько проблем, например долгая загрузка данных с сервера и большая нагрузка на устройство клиента при рендеринге этого списка. Для того, чтобы избежать этого, была использована пагинация. Данные разделены постранично, что уменьшило время получения данных с сервера, а также ускорило процесс рендеринга списка за счёт уменьшения количества элементов на странице.

Для того, чтобы изображения не замедляли загрузку, были использованы такие форматы изображений, как *JPEG* и *WebP*, которые предоставляют эффективные алгоритмы сжатия, позволяющие сохранять изображения с высоким качеством при минимальных размерах файлов.

Экономия ресурсов, связанная с внедрением разработанного программного обеспечения, в основном заключается в сокращении трудоемкости.Применительно к данной работе, при разработке приложения также возможны следующие варианты экономии ресурсов:

* + экономия средств на приобретение и сопровождение программного обеспечения в результате перехода на клиент-серверные технологии, так как можно использовать бесплатные или условно бесплатные пакеты;
  + нет необходимости в установке более современных клиентских компьютеров, т.к. программа не требует значительных вычислительных мощностей и работоспособна на имеющемся оборудовании.

Также важны мероприятия по энергосбережению. Один из самых действенных способов увеличения эффективности использования энергии – применение современных технологий энергосбережения. Технологии энергосбережения не только дают значительное уменьшение расходов на энергетические затраты, но и имеют очевидные экологические плюсы.

Внедрение разработанного ПО с учётом ресурсо- и энергосберегающих мероприятий может привести к экономии ресурсов, снижению энергопотребления и, как следствие, сокращению затрат на энергию и ресурсы. Однако, необходимо учитывать, что каждый случай внедрения ПО уникален, и эффективность ресурсо- и энергосберегающих мероприятий может различаться в зависимости от конкретной ситуации и контекста использования ПО.

# **ЗАКЛЮЧЕНИЕ**

В ходе дипломной работы в соответствии с поставленной задачей было разработано *web*-приложение по учёту и мониторингу выполнения заказов фирмы по производству мебели.

Был проведен анализ предметной области и обзор существующих аналогов, что позволило выявить основные требования и задачи, стоящие перед разрабатываемым приложением. Была разработана функциональная модель задачи автоматизации учета и мониторинга выполнения заказов фирмы по производству мебели, а также структура и состав информационного обеспечения.

В процессе разработки была использована база данных *MS SQL* для хранения информации о мебели и остальных сущностях *web*-приложения, платформа *ASP.NET Core* для разработки *web*-приложения с использованием объектно-ориентированного языка программирования *C#* в среде разработки *Visual Studio Code*. Для доступа данных была использована технология *Entity Framework Core*, которая позволяет создавать запросы к базе данных на языке *LINQ*, фреймворк *Bootstrap* для создания приятного пользовательского интерфейса. Для визуализации *3D*-моделей используется библиотека *JavaScript* [*Three.js*](https://en.wikipedia.org/wiki/Three.js), содержащая набор готовых классов для создания и отображения интерактивной *3D-*графики. Использовался паттерн проектирования *MVC* для разделения логики приложения на отдельные компоненты, фреймворк *ASP.NET Core Identity* для организации аутентификации и авторизации пользователей, обеспечивающий безопасность доступа к функциональности приложения.

Приложение было разработано в соответствии с принципами трехслойной клиент-серверной архитектуры. Трехслойная клиент-серверная архитектура приложения позволяет эффективно разделять логику приложения на отдельные компоненты, улучшая его модульность и обеспечивая более гибкую и масштабируемую разработку.

Таким образом, было создано *web*-приложение с удобным интерфейсом и интуитивно понятными функциями, которое позволит клиентам осуществлять удобный поиск мебели, заказывать мебель из каталога или с заданными параметрами, отслеживать свои заказы, оставлять отзывы, поможет сотрудникам вести учёт заказов и товаров, поможет фирме получить статистику, которая может быть использована для улучшения бизнес-процессов и принятия решений. Все поставленные задачи и цели данной дипломной работы были выполнены в полном объеме.

# **Список использованных источников**

1. Интернет-портал / Белорусский производственно-торговый концерн лесной, деревообрабатывающей и целлюлозно-бумажной промышленности. – Электрон. данные. – Режим доступа: *https://www.bellesbumprom.by*. – Дата доступа: 10.04.2024 год.
2. Интернет-портал / ЗАО «Холдинговая компания «Пинскдрев»» – Электрон. данные. – Режим доступа: *https://pinskdrev.by*. – Дата доступа: 10.04.2024 год.
3. Интернет-портал / ООО «ДИВАН БАЙ». – Электрон. данные. – Режим доступа: *https://www.divan.by.* – Дата доступа: 10.04.2024 год.
4. Интернет-магазин «ask.by». – Электрон. данные. – Режим доступа: *https://aks.by*. – Дата доступа: 10.04.2024 год.
5. Интернет-портал / ООО «Мебель для вас». – Электрон. данные. – Режим доступа: *https://aksamit.by*. – Дата доступа: 10.04.2024 год.
6. Мак-Дональд, М. Создание Web-сайта. Недостающее руководство / М. Мак-Дональд. – 3-е изд. / пер. с англ. – СПб.: БХВ-Петербург, 2013. – 624 с.
7. Шилдт, Г. C# 4.0: полное руководство / пер. с англ. – М.: ООО «И.Д. Вильямс», 2011. – 1056 с.
8. Магдануров, Г. И. *ASP.NET MVC Framework* / Г.И. Магдануров, В. А. Юнев – СПб.: БХВ-Петербург, 2010. – 320 с.
9. Бен-Ган, И. *Microsoft SQL Server* 2012. Основы *T-SQL* / пер. с англ М. А. Райтмана. – Москва: Эксмо, 2015 – 400 с.
10. Смит, Дж. П. *Entity Framework Core* в действии / пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2022. – 690 с.
11. Буч, Г. Язык UML. Руководство пользователя / Буч Г., Рамбо Д., Якобсон И. – 2-е изд., пер. с англ. Мухин Н. – М.: ДМК Пресс. – 496 с.
12. Тарасов, С. В. СУБД для программиста. Базы данных изнутри / Тарасов С. В. – М.: СОЛОН-Пресс, 2015. – 320 с.
13. Кожевников, Е. А. Расчет экономической эффективности разработки программных продуктов: пособие для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» днев. формы обучения / Е. А. Кожевников, Н. В. Ермалинская. – Гомель: ГГТУ им. П. О. Сухого, 2012. – 68 с.
14. ПЭВМ – Электрон. данные. – Режим доступа: *https://www.21vek.by/*

*notebooks/tufgminga15fa506nfhn061asus8878730.html* – Дата доступа 02.06.2024.

1. Базовая ставка в РБ – Электрон. данные. – Режим доступа: *https://www.mintrud.gov.by/by/news-by/view/s-1-janvarja-2024-g-vyrastet-zarabo tnaja-plata-v-bjudzhetnyx-organizatsijax-7654-2023* – Дата доступа 11.05.2024.
2. Об охране труда : Указ Президента Респ. Беларусь № 356-3 : подписан Президентом Респ. Беларусь 23 июня 2008 г. : рег. № 1/1453 от 26 июня 2008 г.
3. Национальный интернет-портал Президента Республики Беларусь. – Минса, 2015. – Электрон. данные. – Режим доступа: [*https://president.gov.by/ru/belarus/social/zashita-naselenija/ohrana-truda*](https://president.gov.by/ru/belarus/social/zashita-naselenija/ohrana-truda). – Дата доступа: 05.06.2024.
4. Национальный стандарт СТБ *ISO* 45001–2020 «Системы менеджмента здоровья и безопасности при профессиональной деятельности. Требования и руководство по применению». – Введ. 25.02.2020. – Минск : БелГИСС : Гос. ком. по стандартизации Респ. Беларусь, 2020. – 44 с.
5. Ткачева, Л.Т. Управление трудоохранной деятельностью : пособие / сост. Л.Т.Ткачева. – Минск: БГАТУ, 2023. – 336 с.
6. Трудовой кодекс Республики Беларусь : Кодекс Республики Беларусь, 26 июля 1999 г., № 296-З : в ред. Закона Респ. Беларусь от 30.06.2022 № 183-З // Нац. реестр правовых актов Республики Беларусь. – 1999. – 80, 2/70.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг классов для работы с базой данных**

namespace WebAppFurniture.DAL.Interfaces

{

public interface IRepository<T> where T : class

{

Task<IEnumerable<T>> GetAllAsync();

Task<T> GetByIdAsync(int id);

Task CreateAsync(T item);

Task UpdateAsync(T item);

Task DeleteAsync(T item);

}

}

namespace WebAppFurniture.DAL.Repositories

{

public abstract class Repository<T> : IRepository<T> where T : class

{

protected readonly DbSet<T> \_dbSet;

protected ApplicationContext \_context;

public Repository(ApplicationContext context)

{

\_context = context;

\_dbSet = context.Set<T>();

}

public abstract Task<T> GetByIdAsync(int id);

public abstract Task<IEnumerable<T>> GetAllAsync();

public async Task CreateAsync(T item)

{

await \_dbSet.AddAsync(item);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(T item)

{

\_dbSet.Update(item);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(T item)

{

\_dbSet.Remove(item);

await \_context.SaveChangesAsync();

}

}

}

namespace WebAppFurniture.DAL.Repositories

{

public class OrderRepository : Repository<Order>, IOrderRepository

{

public OrderRepository(ApplicationContext context) : base(context)

{

}

public override async Task<IEnumerable<Order>> GetAllAsync()

{

return await \_dbSet.AsNoTracking()

.ToListAsync();

}

public async override Task<Order> GetByIdAsync(int id) => await \_dbSet.AsNoTracking()

.FirstOrDefaultAsync(a => a.Id == id);

}

}

namespace WebAppFurniture.DAL.Repositories

{

public class ProductRepository : Repository<Product>, IProductRepository

{

public ProductRepository(ApplicationContext context) : base(context)

{

}

public override async Task<IEnumerable<Product>> GetAllAsync()

{

return await \_dbSet

.AsNoTracking()

.Include(p => p.Orders)

.Include(p => p.WarehouseProducts)

.Include(p => p.ParameterProducts)

.ToListAsync();

}

public async override Task<Product> GetByIdAsync(int id) => await \_dbSet.AsNoTracking()

.Include(t => t.Orders)

.Include(t => t.WarehouseProducts)

.Include(t => t.ParameterProducts)

.FirstOrDefaultAsync(a => a.Id == id);

}

}

namespace WebAppFurniture.DAL.Repositories

{

public class ProviderRepository : Repository<Provider>, IProviderRepository

{

public ProviderRepository(ApplicationContext context) : base(context)

{

}

public override async Task<IEnumerable<Provider>> GetAllAsync()

{

return await \_dbSet.AsNoTracking()

.Include(t => t.Products)

.ToListAsync();

}

public override async Task<Provider> GetByIdAsync(int id) => await \_dbSet.AsNoTracking()

.Include(t => t.Products)

.FirstOrDefaultAsync(a => a.Id == id);

}

}

namespace WebAppFurniture.DAL.Repositories

{

public class ClientRepository : Repository<Client>, IClientRepository

{

public ClientRepository(ApplicationContext context) : base(context)

{

}

public override async Task<IEnumerable<Client>> GetAllAsync()

{

return await \_dbSet.AsNoTracking()

.Include(t => t.Reviews)

.Include(t => t.Orders)

.Include(t => t.Notifications)

.Include(t => t.User)

.ToListAsync();

}

public async override Task<Client> GetByIdAsync(int id) => await \_dbSet.AsNoTracking()

.Include(t => t.Reviews)

.Include(t => t.Orders)

.Include(t => t.Notifications)

.Include(t => t.User)

.FirstOrDefaultAsync(a => a.Id == id);

}

}

amespace WebAppFurniture.DAL.Repositories

{

public class ParameterProductRepository : Repository<ParameterProduct>, IParameterProductRepository

{

public ParameterProductRepository(ApplicationContext context) : base(context)

{

}

public override async Task<IEnumerable<ParameterProduct>> GetAllAsync()

{

return await \_dbSet.AsNoTracking()

.ToListAsync();

}

public override async Task<ParameterProduct> GetByIdAsync(int id)=> await \_dbSet.AsNoTracking()

.FirstOrDefaultAsync(a => a.Id == id);

}

}

namespace WebAppFurniture.DAL.Repositories

{

public class ParameterRepository : Repository<Parameter>, IParameterRepository

{

public ParameterRepository(ApplicationContext context) : base(context)

{

}

public override async Task<IEnumerable<Parameter>> GetAllAsync()

{

return await \_dbSet.AsNoTracking()

.Include((t => t.ParameterProducts))

.ToListAsync();

}

public override async Task<Parameter> GetByIdAsync(int id) => await \_dbSet.AsNoTracking()

.Include((t => t.ParameterProducts))

.FirstOrDefaultAsync(a => a.Id == id);

}

}

namespace WebAppFurniture.DAL.Repositories

{

public class ProductGroupRepository : Repository<ProductGroup>, IProductGroupRepository

{

public ProductGroupRepository(ApplicationContext context) : base(context)

{

}

public override async Task<IEnumerable<ProductGroup>> GetAllAsync()

{

return await \_dbSet.AsNoTracking()

.Include(t => t.Products)

.ToListAsync();

}

public async override Task<ProductGroup> GetByIdAsync(int id) => await \_dbSet.AsNoTracking()

.Include(t => t.Products)

.FirstOrDefaultAsync(a => a.Id == id);

}

}

namespace WebAppFurniture.DAL.Repositories

{

public class WarehouseProductRepository : Repository<WarehouseProduct>, IWarehouseProductRepository

{

public WarehouseProductRepository(ApplicationContext context) : base(context)

{

}

public override async Task<IEnumerable<WarehouseProduct>> GetAllAsync()

{

return await \_dbSet.AsNoTracking().ToListAsync();

}

public override async Task<WarehouseProduct> GetByIdAsync(int id)=> await \_dbSet.AsNoTracking()

.FirstOrDefaultAsync(a => a.Id == id);

}

}

namespace WebAppFurniture.DAL.Repositories

{

public class WarehouseRepository : Repository<Warehouse>, IWarehouseRepository

{

public WarehouseRepository(ApplicationContext context) : base(context)

{

}

public override async Task<IEnumerable<Warehouse>> GetAllAsync()

{

return await \_dbSet.AsNoTracking()

.Include(t => t.WarehouseProducts)

.ToListAsync();

}

public override async Task<Warehouse> GetByIdAsync(int id) => await \_dbSet.AsNoTracking()

.Include(t => t.WarehouseProducts).FirstOrDefaultAsync(a => a.Id == id);

}

}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Листинг классов для организации взаимодействия пользователя с приложением**

namespace WebAppFurniture.BLL.Interfaces

{

public interface IService<T>

{

Task<T> CreateAsync(T entity);

Task<T> UpdateAsync(T entity);

Task<T> DeleteAsync(int id);

Task<T> GetByIdAsync(int id);

Task<IEnumerable<T>> GetAllAsync();

}

}

namespace WebAppFurniture.BLL.Services

{

public class ClientService : IClientService

{

private readonly IClientRepository \_clientRepository;

private readonly IMapper \_mapper;

public ClientService(IMapper mapper, IClientRepository clientRepository)

{

\_mapper = mapper;

\_clientRepository = clientRepository;

}

public async Task<ClientDTO> CreateAsync(ClientDTO entity)

{

var mappedEntity = \_mapper.Map<Client>(entity);

await \_clientRepository.CreateAsync(mappedEntity);

return entity;

}

public async Task<ClientDTO> DeleteAsync(int id)

{

var existingEntity = await \_clientRepository.GetByIdAsync(id);

if (existingEntity == null)

{

throw new ArgumentException($" not found.");

}

await \_clientRepository.DeleteAsync(\_mapper.Map<Client>(existingEntity));

return \_mapper.Map<ClientDTO>(existingEntity);

}

public async Task<IEnumerable<ClientDTO>> GetAllAsync()

{

var allClients = \_mapper.Map<IEnumerable<ClientDTO>>(await \_clientRepository.GetAllAsync());

return allClients;

}

public async Task<ClientDTO> GetByIdAsync(int id)

{

var client = \_mapper.Map<ClientDTO>(await \_clientRepository.GetByIdAsync(id));

return client;

}

public async Task<ClientDTO> GetClientByUser(ClientDTO entity)

{

var allClients = await \_clientRepository.GetAllAsync();

var client = allClients.FirstOrDefault(c => c.User.Email == entity.User.Email);

var mapped = \_mapper.Map<ClientDTO>(client);

return mapped;

}

public async Task<ClientDTO> GetClientByUserId(string id)

{

var allClients = await \_clientRepository.GetAllAsync();

var client = allClients.FirstOrDefault(t => t.User.Id == id);

var result = \_mapper.Map<ClientDTO>(client);

return result;

}

public async Task<ClientDTO> UpdateAsync(ClientDTO entity)

{

var existing = await \_clientRepository.GetByIdAsync(entity.Id);

if (existing == null)

{

throw new ArgumentException("Client not found");

}

existing.Surname = entity.Surname;

existing.Phone = entity.Phone;

existing.Address = entity.Address;

existing.Picture = entity.Picture;

existing.User = entity.User;

existing.Reviews = null;

existing.Orders = null;

existing.Notifications = null;

try

{

await \_clientRepository.UpdateAsync(existing);

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

return entity;

}

}

}

namespace WebAppFurniture.BLL.Services

{

public class OrderService : IOrderService

{

private readonly IOrderRepository \_orderRepository;

private readonly IProductRepository \_productRepository;

private readonly IProductGroupRepository \_productGroupRepository;

private readonly IProviderRepository \_providerRepository;

private readonly IMapper \_mapper;

public OrderService(IMapper mapper, IOrderRepository prderRepository,

IProductRepository productRepository, IProductGroupRepository productGroupRepository, IProviderRepository providerRepository)

{

\_mapper = mapper;

\_orderRepository = prderRepository;

\_productRepository = productRepository;

\_productGroupRepository = productGroupRepository;

\_providerRepository = providerRepository;

}

public async Task<OrderDTO> CreateAsync(OrderDTO entity)

{

var mappedEntity = \_mapper.Map<Order>(entity);

await \_orderRepository.CreateAsync(mappedEntity);

return entity;

}

public async Task<OrderDTO> DeleteAsync(int id)

{

var existingEntity = await \_orderRepository.GetByIdAsync(id);

if (existingEntity == null)

{

throw new ArgumentException($"not found.");

}

await \_orderRepository.DeleteAsync(\_mapper.Map<Order>(existingEntity));

return \_mapper.Map<OrderDTO>(existingEntity);

}

public async Task<IEnumerable<OrderDTO>> GetAllAsync()

{

var mapped = \_mapper.Map<IEnumerable<OrderDTO>>(await \_orderRepository.GetAllAsync());

return mapped;

}

public async Task<OrderDTO> GetByIdAsync(int id)

{

var entity = \_mapper.Map<OrderDTO>(await \_orderRepository.GetByIdAsync(id));

if (entity == null)

{

throw new ArgumentException("not found");

}

var mapped = \_mapper.Map<OrderDTO>(entity);

return mapped;

}

public async Task<IEnumerable<CategorySalesStatisticsDTO>> GetCategorySalesStatisticsAsync(DateTime startDate, DateTime endDate)

{

var mappedOrders = \_mapper.Map<IEnumerable<Order>>(await \_orderRepository.GetAllAsync());

foreach (var order in mappedOrders)

{

order.Product = await \_productRepository.GetByIdAsync(order.ProductId);

order.Product.Provider = await \_providerRepository.GetByIdAsync(order.Product.ProviderId);

order.Product.ProductGroup = await \_productGroupRepository.GetByIdAsync(order.Product.ProductGroupId);

}

var categorySalesStatistics = mappedOrders

.Where(o => o.Date >= startDate && o.Date <= endDate && o.Status != "Отменен" && o.Product.Type != "под заказ")

.GroupBy(o => o.Product.ProductGroup.Name)

.Select(g => new CategorySalesStatisticsDTO

{

CategoryName = g.Key,

TotalSales = g.Count()

})

.OrderByDescending(c => c.TotalSales);

return categorySalesStatistics;

}

public async Task<IEnumerable<CanceledSalesStatisticsDTO>> GetCanceledSalesStatisticsAsync(DateTime startDate, DateTime endDate)

{

var orders = \_mapper.Map<IEnumerable<Order>>(await \_orderRepository.GetAllAsync());

var canceledOrders = orders.Where(o => o.Status == "Отменен");

foreach (var order in canceledOrders)

{

order.Product = await \_productRepository.GetByIdAsync(order.ProductId);

order.Product.Provider = await \_providerRepository.GetByIdAsync(order.Product.ProviderId);

order.Product.ProductGroup = await \_productGroupRepository.GetByIdAsync(order.Product.ProductGroupId);

}

var canceledSalesStatistics = canceledOrders

.Where(o => o.Date >= startDate && o.Date <= endDate)

.GroupBy(o => new { o.Product.Name, ProductGroupName = o.Product.ProductGroup?.Name })

.Select(g => new CanceledSalesStatisticsDTO

{

ProductName = g.Key.Name,

ProductGroupName = g.Key.ProductGroupName,

CountCanceled = g.Count()

});

return canceledSalesStatistics;

}

public async Task<double> GetTotalCostStatisticsAsync(DateTime startDate, DateTime endDate)

{

var orders = \_mapper.Map<IEnumerable<Order>>(await \_orderRepository.GetAllAsync());

foreach (var order in orders) order.Product = await \_productRepository.GetByIdAsync(order.ProductId);

double totalCost = orders

.Where(order => order.Date >= startDate && order.Date <= endDate && order.Product.Type!="под заказ")

.Sum(order => order.TotalCost);

return totalCost;

}

public async Task<IEnumerable<OrderDTO>> GetOrdersByClientIdAsync(int id)

{

var allOrders = \_mapper.Map<IEnumerable<OrderDTO>>(await \_orderRepository.GetAllAsync());

return allOrders.Where(o => o.ClientId == id);

}

public async Task<IEnumerable<ProviderSalesStatisticsDTO>> GetProviderSalesStatisticsAsync(DateTime StartDate, DateTime EndDate)

{

var mappedOrders = \_mapper.Map<IEnumerable<Order>>(await \_orderRepository.GetAllAsync());

foreach (var order in mappedOrders)

{

var product = await \_productRepository.GetByIdAsync(order.ProductId);

order.Product = product;

var provider = await \_providerRepository.GetByIdAsync(order.Product.ProviderId);

order.Product.Provider = provider;

var productgroup = await \_productGroupRepository.GetByIdAsync(order.Product.ProductGroupId);

order.Product.ProductGroup = productgroup;

}

var providerSalesStatistics = mappedOrders

.Where(o => o.Date >= StartDate && o.Date <= EndDate)

.GroupBy(o => o.Product.Provider.Name)

.Select(g => new ProviderSalesStatisticsDTO

{

ProviderName = g.Key,

TotalSales = (decimal)g.Sum(o => o.TotalCost)

})

.OrderByDescending(c => c.TotalSales);

return providerSalesStatistics;

}

public async Task<OrderDTO> UpdateAsync(OrderDTO entity)

{

var existing = await \_orderRepository.GetByIdAsync(entity.Id);

if (existing == null)

{

throw new ArgumentException("not found");

}

existing.ClientId = entity.ClientId;

existing.ProductId = entity.ProductId;

existing.Date = entity.Date;

existing.Status = entity.Status;

existing.TotalCost=entity.TotalCost;

existing.ProgressCount = entity.ProgressCount;

try

{

await \_orderRepository.UpdateAsync(existing);

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

return \_mapper.Map<OrderDTO>(existing);

}

}

}

namespace WebAppFurniture.BLL.Services

{

public class NotificationService : INotificationService

{

private readonly INotificationRepository \_notificationRepository;

private readonly IMapper \_mapper;

public NotificationService(IMapper mapper, INotificationRepository notificationRepository)

{

\_mapper = mapper;

\_notificationRepository = notificationRepository;

}

public async Task<NotificationDTO> CreateAsync(NotificationDTO entity)

{

var mappedEntity = \_mapper.Map<Notification>(entity);

await \_notificationRepository.CreateAsync(mappedEntity);

return entity;

}

public async Task<NotificationDTO> DeleteAsync(int id)

{

var existingEntity = await \_notificationRepository.GetByIdAsync(id);

if (existingEntity == null)

{

throw new ArgumentException($"Notification not found.");

}

await \_notificationRepository.DeleteAsync(\_mapper.Map<Notification>(existingEntity));

return \_mapper.Map<NotificationDTO>(existingEntity);

}

public async Task<IEnumerable<NotificationDTO>> GetAllAsync()

{

var mapped = \_mapper.Map<IEnumerable<NotificationDTO>>(await \_notificationRepository.GetAllAsync());

return mapped;

}

public async Task<NotificationDTO> GetByIdAsync(int id)

{

var entity = \_mapper.Map<NotificationDTO>(await \_notificationRepository.GetByIdAsync(id));

if (entity == null)

{

throw new ArgumentException("not found");

}

var mapped = \_mapper.Map<NotificationDTO>(entity);

return mapped;

}

public async Task<IEnumerable<NotificationDTO>> GetNotificationsByClientIdAsync(int id)

{

var all = \_mapper.Map<IEnumerable<NotificationDTO>>(await \_notificationRepository.GetAllAsync());

return all.Where(o => o.ClientId == id);

}

public async Task<IEnumerable<NotificationDTO>> MarkAllAsReadAsync(int clientid)

{

var all = \_mapper.Map<IEnumerable<NotificationDTO>>(await \_notificationRepository.GetAllAsync());

var notifications = all.Where(o => o.ClientId == clientid);

foreach (var notification in notifications)

{

if (notification.Status == "Непрочитано")

{

notification.Status = "Прочитано";

await \_notificationRepository.UpdateAsync(\_mapper.Map<Notification>(notification));

}

}

return notifications;

}

public async Task<NotificationDTO> UpdateAsync(NotificationDTO entity)

{

var existingEntity = await \_notificationRepository.GetByIdAsync(entity.Id);

if (existingEntity == null) throw new ArgumentException($"{entity.Id} not found.");

await \_notificationRepository.UpdateAsync(existingEntity);

return \_mapper.Map<NotificationDTO>(existingEntity);

}

}}

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Листинг классов контроллеров**

namespace WebAppFurnitureOrders.Controllers

{

public class AccountController : Controller

{

private readonly UserManager<IdentityUser> \_userManager;

private readonly SignInManager<IdentityUser> \_signInManager;

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly IClientService \_clientService;

private readonly INotificationService \_notificationService;

private readonly IWebHostEnvironment \_hostEnvironment;

private readonly IMapper \_mapper;

public AccountController(IMapper mapper, RoleManager<IdentityRole> roleManager,INotificationService notificationService,

IWebHostEnvironment hostEnvironment,IClientService clientService, UserManager<IdentityUser> userManager, SignInManager<IdentityUser> signInManager)

{

\_mapper = mapper;

\_userManager = userManager;

\_signInManager = signInManager;

\_clientService = clientService;

\_hostEnvironment = hostEnvironment;

\_roleManager = roleManager;

\_notificationService = notificationService;

}

[HttpPost]

public async Task<IActionResult> Register(RegisterUser model)

{

if (ModelState.IsValid)

{

var existingUser = await \_userManager.FindByEmailAsync(model.Email);

var emailPattern = @"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$";

var isValidEmail = Regex.IsMatch(model.Email, emailPattern);

if (existingUser != null)

{

ModelState.AddModelError("Email", "Пользователь с таким email уже существует");

var errorss = ModelState.Values.SelectMany(v => v.Errors.Select(e => e.ErrorMessage));

TempData["ErrorsReg"] = errorss;

return View(model);

}

if (!isValidEmail)

{

ModelState.AddModelError("Email", "Неправильный формат email");

var errorss = ModelState.Values.SelectMany(v => v.Errors.Select(e => e.ErrorMessage));

TempData["ErrorsReg"] = errorss;

return View(model);

}

IdentityUser user = new IdentityUser { UserName = model.Email, Email = model.Email };

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, false);

ClientDTO client = new ClientDTO() { Phone = " ", User = user, Address=" " ,Picture="default.png", Surname=" "};

var clientResult = await \_clientService.CreateAsync(client);

var roleExists = await \_roleManager.RoleExistsAsync("Клиент");

if (!roleExists)

{

var role = new IdentityRole("Клиент");

await \_roleManager.CreateAsync(role);

}

await \_userManager.AddToRoleAsync(user, "Клиент");

var newClient = await \_clientService.GetClientByUser(clientResult);

NotificationDTO notification = new NotificationDTO() { Message="Добро пожаловать!",Status="Непрочитано",ClientId = newClient.Id,Date=DateTime.Now};

await \_notificationService.CreateAsync(notification);

return RedirectToAction("Catalog", "Home");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

var errors = ModelState.Values.SelectMany(v => v.Errors.Select(e => e.ErrorMessage));

TempData["ErrorsReg"] = errors;

return View(model);

}

[HttpPost]

public async Task<IActionResult> LogIn(string email, string password)

{

if (ModelState.IsValid)

{

var existingUser = await \_userManager.FindByEmailAsync(email);

if (existingUser == null)

{

ModelState.AddModelError("Email", "Пользователя с таким email не существует");

return View();

}

var result = await \_signInManager.PasswordSignInAsync(email, password, true, false);

if (!result.Succeeded)

{

ModelState.AddModelError("WrongEmailOrPass", "Неверный пароль");

return View();

}

return RedirectToAction("Catalog", "Home");

}

return View();

}

public async Task<IActionResult> LogOutAsync()

{

await \_signInManager.SignOutAsync();

return RedirectToAction("Catalog", "Home");

}

public async Task<IActionResult> EditProfileAsync()

{

var user = await \_userManager.GetUserAsync(User);

var client = \_mapper.Map <ClientDTO>( await \_clientService.GetClientByUserId(user.Id));

var result = new ClientModel

{

Email = user.Email,

Surname = client.Surname ?? "",

Address = client.Address ?? "",

Phone = client.Phone ?? "",

Picture = client.Picture ?? "",

};

return View(result);

}

[HttpPost]

public async Task<IActionResult> EditProfileAsync(ClientModel model)

{

var user = await \_userManager.GetUserAsync(User);

var client = await \_clientService.GetClientByUserId(user.Id);

if (!string.IsNullOrEmpty(model.CurrentPassword))

{

var isPasswordValid = await IsPasswordValid(model.CurrentPassword, user.Id);

if (isPasswordValid)

{

if (model.ImageFile != null)

{

model.Picture = null;

string wwwRootPath = \_hostEnvironment.WebRootPath;

string fileName = Path.GetFileNameWithoutExtension(model.ImageFile.FileName);

string extension = Path.GetExtension(model.ImageFile.FileName);

model.Picture = fileName = fileName + DateTime.Now.ToString("yymmssfff") + extension;

string path = Path.Combine(wwwRootPath + "/images/avatar", fileName);

using (var fileStream = new FileStream(path, FileMode.Create))

{

await model.ImageFile.CopyToAsync(fileStream);

}

}

else

{

model.Picture = client.Picture;

}

if (!string.IsNullOrEmpty(model.Phone) || !string.IsNullOrEmpty(model.Surname))

{

var phoneRegex = new Regex(@"^\+?\d{1,3}?\d{9}$");

if (!phoneRegex.IsMatch(model.Phone))

{

ModelState.AddModelError("Phone", "Неправильный формат номера телефона.");

}

else

{

client = await \_clientService.GetClientByUserId(user.Id);

var updated = new ClientDTO

{

Id = client.Id,

Surname = model.Surname ?? "",

Address = model.Address ?? "",

Phone = model.Phone ?? "",

Picture = model.Picture ?? "",

};

await \_clientService.UpdateAsync(updated);

if (user != null && !string.IsNullOrEmpty(model.NewPassword))

{

var result = await \_signInManager.PasswordSignInAsync(user.UserName, model.CurrentPassword, isPersistent: false, lockoutOnFailure: false);

if (result.Succeeded)

{

if (model.NewPassword == model.ConfirmPassword)

{

var changePasswordResult = await \_userManager.ChangePasswordAsync(user, model.CurrentPassword, model.NewPassword);

if (changePasswordResult.Succeeded)

{

TempData["SuccessMessage"] = "Пароль успешно обновлен.";

}

else ModelState.AddModelError("ConfirmPassword", "Ошибка.");

}

else ModelState.AddModelError("ConfirmPassword", "Пароль и его подтверждение не совпадают.");

}

}

else TempData["SuccessMessage"] = "Данные обновлены.";

}

}

else { ModelState.AddModelError("Phone", "Введите номер телефона и ФИО."); }

}

else ModelState.AddModelError("CurrentPassword", "Неверный пароль.");

}

else { ModelState.AddModelError("CurrentPassword", "Введите пароль."); }

return View(model);

}

public async Task<bool> IsPasswordValid(string currentPassword, string userId)

{

var user = await \_userManager.FindByIdAsync(userId);

if (user != null)

{

var result = await \_signInManager.CheckPasswordSignInAsync(user, currentPassword, false);

return result.Succeeded;

}

return false;

}

}

}

namespace WebAppFurnitureOrders.Controllers

{

public class NotificationController : Controller

{

private readonly INotificationService \_notificationService;

private readonly UserManager<IdentityUser> \_userManager;

private readonly IClientService \_clientService;

private readonly IMapper \_mapper;

public NotificationController(INotificationService notificationService, IClientService clientService, UserManager<IdentityUser> userManager, IMapper mapper)

{

\_notificationService = notificationService;

\_clientService = clientService;

\_userManager = userManager;

\_mapper = mapper;

}

public async Task<IActionResult> GetNotifications()

{

var user = await \_userManager.GetUserAsync(User);

var client = \_mapper.Map<ClientDTO>(await \_clientService.GetClientByUserId(user.Id));

var notifications = await \_notificationService.GetNotificationsByClientIdAsync(client.Id);

return Json(notifications);

}

public async Task<IActionResult> CheckUnreadNotifications()

{

var user = await \_userManager.GetUserAsync(User);

var client = \_mapper.Map<ClientDTO>(await \_clientService.GetClientByUserId(user.Id));

var notifications = await \_notificationService.GetNotificationsByClientIdAsync(client.Id);

var hasUnread = notifications.Any(n => n.Status == "Непрочитано");

return Json(new { hasUnread });

}

[HttpPost]

public async Task<IActionResult> MarkAllAsRead()

{

var user = await \_userManager.GetUserAsync(User);

var client = \_mapper.Map<ClientDTO>(await \_clientService.GetClientByUserId(user.Id));

await \_notificationService.MarkAllAsReadAsync(client.Id);

return Ok();

}

}

}

namespace WebAppFurnitureOrders.Controllers

{

public class ReviewController : Controller

{

private readonly IReviewService \_reviewService;

private readonly IMapper \_mapper;

private readonly UserManager<IdentityUser> \_userManager;

private readonly SignInManager<IdentityUser> \_signInManager;

private readonly IClientService \_clientService;

public ReviewController(IMapper mapper, IReviewService reviewService, UserManager<IdentityUser> userManager, SignInManager<IdentityUser> signInManager, IClientService clientService)

{

\_mapper = mapper;

\_reviewService = reviewService;

\_userManager = userManager;

\_signInManager = signInManager;

\_clientService = clientService;

}

public async Task<IActionResult> AllReviewsAsync()

{

var reviews = \_mapper.Map<List<ReviewModel>>(await \_reviewService.GetAllAsync());

foreach (var review in reviews)

{

var client = \_mapper.Map<ClientModel>(await \_clientService.GetByIdAsync(review.ClientId));

review.Client = client;

}

reviews.Reverse();

return View(reviews);

}

[HttpPost]

public async Task<IActionResult> AddReview(ReviewModel model)

{

var user = await \_userManager.GetUserAsync(User);

var client = await \_clientService.GetClientByUserId(user.Id);

if (model.Comment==null || model.Rating==0 || string.IsNullOrWhiteSpace(client.Surname))

{

var reviews = \_mapper.Map<List<ReviewModel>>(await \_reviewService.GetAllAsync());

foreach (var review in reviews)

{

var userr = await \_userManager.GetUserAsync(User);

var clientt = \_mapper.Map<ClientModel>(await \_clientService.GetClientByUserId(userr.Id));

review.Client = clientt;

}

if (string.IsNullOrWhiteSpace(client.Surname))

{

TempData["ErrorNull"] = "Заполните ФИО в профиле, чтобы оставить отзыв";

}

return View("AllReviews", reviews);

}

TempData["ErrorNull"] = null;

var cl = \_mapper.Map<ClientDTO> (client);

if (client != null)

{

ReviewDTO review = new ReviewDTO()

{

ClientId = cl.Id,

Comment = model.Comment,

Rating=model.Rating,

};

await \_reviewService.CreateAsync(review);

}

return RedirectToAction("AllReviews");

}

}

}

namespace WebAppFurnitureOrders.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

private readonly IWarehouseService \_warehouseService;

private readonly IProductService \_productService;

private readonly IWarehouseProductService \_warehouseProductService;

private readonly INotificationService \_notificationService;

private readonly IProviderService \_providerService;

private readonly IProductGroupService \_productGroupService;

private readonly UserManager<IdentityUser> \_userManager;

private readonly IClientService \_clientService;

private readonly IOrderService \_orderService;

private readonly IMapper \_mapper;

public HomeController(ILogger<HomeController> logger, IMapper mapper,

IProductService productService, IWarehouseProductService warehouseProductService, IWarehouseService warehouseService,

IProductGroupService productGroupService, IProviderService providerService, UserManager<IdentityUser> userManager, IClientService clientService, IOrderService orderService, INotificationService notificationService)

{

\_logger = logger;

\_mapper = mapper;

\_productService = productService;

\_warehouseProductService = warehouseProductService;

\_warehouseService = warehouseService;

\_productGroupService = productGroupService;

\_providerService = providerService;

\_userManager = userManager;

\_clientService = clientService;

\_orderService = orderService;

\_notificationService = notificationService;

}

public async Task<IActionResult> Catalog()

{

var warehouseProducts = \_mapper.Map<List<WarehouseProductModel>>(await \_warehouseProductService.GetAllAsync());

var products = new List<ProductViewModel>();

var productss = \_mapper.Map<List<ProductModel>>(await \_productService.GetAllCatalogAsync());

await Output(products, productss, warehouseProducts);

return View(products);

}

private async Task Output(List<ProductViewModel> products, List<ProductModel> pro, List<WarehouseProductModel> warehouseProducts)

{

foreach (var p in pro)

{

var provider = \_mapper.Map<ProviderModel>(await \_providerService.GetByIdAsync(p.ProviderId));

var group = \_mapper.Map<ProductGroupModel>(await \_productGroupService.GetByIdAsync(p.ProductGroupId));

p.Provider = provider;

p.ProductGroup = group;

var associatedWarehouseProducts = warehouseProducts.Where(wp => wp.ProductId == p.Id).ToList();

var images = new List<string>();

if (associatedWarehouseProducts.Count > 0)

{

foreach (var warehouseProduct in associatedWarehouseProducts)

{

var warehouse = \_mapper.Map<WarehouseModel>(await \_warehouseService.GetByIdAsync(warehouseProduct.WarehouseId));

var imageUrl = warehouse.ImageUrl;

images.Add(imageUrl);

}

var productViewModel = new ProductViewModel

{

Id = p.Id,

Name = p.Name,

Cost = (double)p.Cost,

Type = p.Type,

ProductGroupId = p.ProductGroupId,

ProductGroup = p.ProductGroup,

ProviderId = p.Provider.Id,

Provider = p.Provider,

ImageUrls = images

};

products.Add(productViewModel);

}

}

}

public async Task<IActionResult> Filtered(string furnitureType)

{

var warehouseProducts = \_mapper.Map<List<WarehouseProductModel>>(await \_warehouseProductService.GetAllAsync());

var products = new List<ProductViewModel>();

var productss = \_mapper.Map<List<ProductModel>>(await \_productService.GetAllFilteredAsync(furnitureType));

await Output(products, productss, warehouseProducts);

if (products != null)

{

ViewBag.Results = products;

}

return View("Catalog", products);

}

public async Task<IActionResult> Search(string searchText)

{

var warehouseProducts = \_mapper.Map<List<WarehouseProductModel>>(await \_warehouseProductService.GetAllAsync());

var products = new List<ProductViewModel>();

var productss = \_mapper.Map<List<ProductModel>>(await \_productService.GetAllSearchedAsync(searchText));

await Output(products, productss, warehouseProducts);

if (products != null) ViewBag.Results = products;

return View("Catalog", products);

}

public async Task<IActionResult> Details(int productId)

{

var product = \_mapper.Map<ProductModel>(await \_productService.GetByIdAsync(productId));

var warehouseProducts = \_mapper.Map<List<WarehouseProductModel>>(await \_warehouseProductService.GetAllAsync());

var provider = \_mapper.Map<ProviderModel>(await \_providerService.GetByIdAsync(product.ProviderId));

var group = \_mapper.Map<ProductGroupModel>(await \_productGroupService.GetByIdAsync(product.ProductGroupId));

var associatedWarehouseProducts = warehouseProducts.Where(wp => wp.ProductId == product.Id).ToList();

var images = new List<string>();

var colors = new List<string>();

foreach (var warehouseProduct in associatedWarehouseProducts)

{

var warehouse = \_mapper.Map<WarehouseModel>(await \_warehouseService.GetByIdAsync(warehouseProduct.WarehouseId));

var imageUrl = warehouse.ImageUrl;

var color = warehouse.Color;

images.Add(imageUrl);

colors.Add(color);

}

var productViewModel = new ProductViewModel

{

Id = product.Id,

Name = product.Name,

Description= product.Description,

Cost = (double)product.Cost,

Type = product.Type,

ProductGroupId = product.ProductGroupId,

ProviderId = product.ProviderId,

ProductGroup = group,

Provider = provider,

ImageUrls = images,

Colors = colors,

};

return View(productViewModel);

}

public async Task<IActionResult> OrderProduct(OrderViewModel model)

{

var product = await \_productService.GetByIdAsync(model.ProductId);

var user = await \_userManager.GetUserAsync(User);

var client = \_mapper.Map<ClientDTO>(await \_clientService.GetClientByUserId(user.Id));

if (string.IsNullOrWhiteSpace(client.Phone) || string.IsNullOrWhiteSpace(client.Surname))

{

return Json(new { success = false, errorMessage = "Заполните поля ФИО и телефона в профиле" });

}

var order = new OrderModel

{

Date = DateTime.Now,

ProgressCount = 0,

TotalCost = model.TotalCost,

Status = model.Status,

ClientId = client.Id,

ProductId = model.ProductId,

};

var orderDto = \_mapper.Map<OrderDTO>(order);

await \_orderService.CreateAsync(orderDto);

@model.Product = \_mapper.Map<ProductModel>(await \_productService.GetByIdAsync(@model.ProductId));

@model.Product.ProductGroup = \_mapper.Map<ProductGroupModel>(await \_productGroupService.GetByIdAsync(@model.Product.ProductGroupId));

var productgroup = @model.Product.ProductGroup.Name;

var name = @model.Product.Name;

NotificationDTO notification = new NotificationDTO() { Message = "Вы заказали " + productgroup + " "+name, Status = "Непрочитано", ClientId = client.Id ,Date=DateTime.Now };

await \_notificationService.CreateAsync(notification);

return Json(new { success = true });

}

}

}