ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

**«ТОМСКИЙ ЭКОНОМИКО-ПРОМЫШЛЕННЫЙ КОЛЛЕДЖ»**

К ЗАЩИТЕ ДОПУЩЕН

Директор ОГБПОУ «ТЭПК»

\_\_\_\_\_\_\_\_\_\_\_\_ / О.Н. Пояркова

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА REST API СЕРВЕРА НА ФРЕЙМВОРКЕ LARAVEL И САЙТА «ТАЙНЫ ВОСТОКА» НА VUE**

КОМПЛЕКСНЫЙ ДИПЛОМНЫЙ ПРОЕКТ

*Специальность 09.02.07 Информационные системы и программирование*

Выполнили:

Студенты IV курса группы 1912с

\_\_\_\_\_\_\_\_\_\_\_\_ / Федоров А.А.

\_\_\_\_\_\_\_\_\_\_\_\_ / Рюгин А.И.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Проверили:

Руководители КДП

\_\_\_\_\_\_\_\_\_\_\_\_ / Уляхин В.А.

\_\_\_\_\_\_\_\_\_\_\_\_ / Летягина И.Е.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 4](#_Toc201039248)

[ТЕХНОЛОГИИ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 5](#_Toc201039249)

[Используемые языки программирования и фреймворки 5](#_Toc201039250)

[Используемое программное обеспечение 7](#_Toc201039251)

[ГЛАВА 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ ИНФОРМАЦИОННОГО ПРОДУКТА 8](#_Toc201039252)

[Описание проекта и задач 8](#_Toc201039253)

[Ролевая модель 10](#_Toc201039254)

[Специфические требования к API 12](#_Toc201039255)

[Разработка прототипа пользовательского интерфейса 44](#_Toc201039256)

[ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ 53](#_Toc201039257)

[Нормализация данных 53](#_Toc201039258)

[Концептуальная модель данных 54](#_Toc201039259)

[Логическая модель данных 55](#_Toc201039260)

[Диаграмма базы данных 57](#_Toc201039261)

[ГЛАВА 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ФУНКЦИОНАЛА ИНФОРМАЦИОННОГО ПРОДУКТА 58](#_Toc201039262)

[Установка и настройка фреймворка Laravel12 58](#_Toc201039263)

[Создание миграций 59](#_Toc201039264)

[Создание исключений 62](#_Toc201039265)

[Создание моделей 63](#_Toc201039266)

[Создание контроллеров 68](#_Toc201039267)

[Валидация запроса формы 87](#_Toc201039268)

[Маршрутизация 93](#_Toc201039269)

[ГЛАВА 4. АВТОТЕСТИРОВАНИЕ API 95](#_Toc201039270)

[401 Unauthorized 95](#_Toc201039271)

[403 Forbidden 96](#_Toc201039272)

[404 Not Found 96](#_Toc201039273)

[200 OK 97](#_Toc201039274)

[201 Created 98](#_Toc201039275)

[ГЛАВА 5. РЕАЛИЗАЦИЯ ФУНКЦИОНАЛА ВЕБ-ПРИЛОЖЕНИЯ 99](#_Toc201039276)

[Реализация API и вспомогательных модулей 99](#_Toc201039277)

[Реализация управления состоянием (Vuex Store) 100](#_Toc201039278)

[Реализация пользовательского интерфейса 103](#_Toc201039279)

[ГЛАВА 6. ТЕСТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ 202](#_Toc201039280)

[ГЛАВА 7. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 206](#_Toc201039281)

[ГЛАВА 8. РУКОВОДСТВО МЕНЕДЖЕРА 208](#_Toc201039282)

[ГЛАВА 9. РУКОВОДСТВО МЕНЕДЖЕРА 210](#_Toc201039283)

[ЗАКЛЮЧЕНИЕ 212](#_Toc201039284)

[СПИСОК ЛИТЕРАТУРЫ 214](#_Toc201039285)

# ВВЕДЕНИЕ

Актуальность проекта:

Электронная коммерция стремительно развивается, предлагая пользователям удобные способы совершения покупок в интернете. Восточная мода, включающая японскую, корейскую и китайскую одежду, становится всё более популярной благодаря уникальному дизайну и высоким стандартам качества. Однако на рынке представлено ограниченное количество специализированных магазинов, предлагающих широкий ассортимент восточной одежды с удобной системой заказов и доставки.

Разработка интернет-магазина «Тайны Востока», ориентированного на продажу одежды из стран Азии, актуальна по следующим причинам:

* Повышенный интерес к азиатскому стилю: Восточная одежда набирает популярность среди ценителей уникального дизайна и современных трендов.
* Рост онлайн-покупок: Все больше пользователей предпочитают удобные интернет-магазины с продуманной системой доставки.
* Оптимизация логистики: Внедрение системы пунктов выдачи заказов позволяет снизить затраты на доставку и ускорить получение товаров.
* Цифровая трансформация бизнеса: Интеграция REST API и автоматизация бизнес-процессов обеспечивают удобное управление заказами, товарами и взаимодействием с клиентами.

Целью дипломного проекта является проектирование и разработка REST API сервера на фреймворке Laravel и сайта «тайны востока» на Vue.

Задачи проекта:

1. Разработка REST API на Laravel для обработки заказов, управления корзиной, пользователями и товарами.
2. Создание веб-интерфейса на Vue, обеспечивающего адаптивность и удобство использования.
3. Реализация механизма аутентификации и личного кабинета с историей заказов.
4. Интеграция пунктов выдачи с выбором удобного места для получения заказов.
5. Разработка системы отзывов для оценки товаров пользователями.

**Распределение задач в проекте**

Разработка системы осуществлялась в тесном взаимодействии двух студентов, каждый из которых отвечал за ключевые направления проекта:

**Рюгин Алексей Иванович:**

* Проектирование и разработка backend-части системы.
* Создание архитектуры базы данных и работа с СУБД.
* Разработка API для взаимодействия между компонентами системы.
* Подготовка всей технической документации (включая ТЗ).
* Тестирование backend-компонентов.

**Федоров Алексей Алексеевич:**

* Разработка пользовательского интерфейса (frontend).
* Создание веб-приложения для покупок одежды из восточных стран.
* UI/UX-проектирование.
* Вёрстка и реализация клиентской логики.
* Тестирование клиентской части.

ТЕХНОЛОГИИ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

## Используемые языки программирования и фреймворки

PHP - это широко используемый язык программирования общего назначения, который применяется в основном для разработки веб-приложений и веб-серверных скриптов. PHP обычно встраивается непосредственно в HTML код, и его скрипты выполняются на сервере, что позволяет генерировать динамический контент на веб-страницах.

Основные характеристики PHP включают в себя:

* Простота использования: PHP имеет синтаксис, который легко читать и понимать, что делает его доступным для новичков.
* Гибкость: PHP поддерживает различные парадигмы программирования, включая процедурное и объектно-ориентированное программирование, что позволяет разработчикам выбирать подход, который лучше всего подходит для их задач.
* Широкая поддержка: PHP является одним из наиболее популярных языков программирования для веб-разработки и имеет обширное сообщество разработчиков, библиотек и фреймворков.
* Интеграция с веб-серверами: PHP легко интегрируется с различными веб-серверами, такими как Apache, Nginx и другие.
* Динамическое создание контента: PHP позволяет создавать динамические веб-страницы, которые могут меняться в зависимости от пользовательского ввода, данных из базы данных и других факторов.
* Базы данных: PHP обладает встроенной поддержкой для работы с различными системами управления базами данных, такими как MySQL, PostgreSQL, SQLite и другими.
* PHP используется для создания широкого спектра веб-приложений, включая блоги, интернет-магазины, корпоративные веб-сайты, социальные сети и многое другое. Он остается одним из наиболее популярных и распространенных языков программирования для веб-разработки благодаря своей простоте, гибкости и широкой поддержке сообщества.

SQL - это специализированный язык, который используется для работы с реляционными базами данных. Он предоставляет набор команд и инструкций для управления данными в базах данных, таких как добавление, удаление, изменение и извлечение данных.

Основные характеристики SQL:

* Декларативный язык: SQL является декларативным языком, что означает, что вы определяете, что вы хотите сделать с данными, а не как это делать. Вы описываете желаемый результат, а не последовательность действий.
* Манипуляция данными: SQL предоставляет команды для манипуляции данными в базе данных. Например, с помощью SQL вы можете создавать новые таблицы, добавлять новые записи, обновлять существующие записи и удалять записи.
* Язык запросов: для извлечения данных из базы данных. Вы можете создавать запросы, чтобы выбирать данные из одной или нескольких таблиц, фильтровать данные, сортировать их и группировать.

Laravel - это бесплатный и с открытым исходным кодом фреймворк для создания веб-приложений на языке программирования PHP. Он предоставляет разработчикам инструменты и структуру для быстрой и эффективной разработки веб-приложений, используя современные практики разработки и популярные шаблоны проектирования.

Основные характеристики Laravel:

* Элегантный синтаксис: Laravel предлагает простой и выразительный синтаксис, который делает код легким для чтения и понимания.
* Мощные функции: Laravel обладает богатым набором функций, включая маршрутизацию, контроллеры, валидацию данных, работу с базой данных, аутентификацию пользователей, кэширование, отправку электронной почты и многое другое.
* Быстрый старт: Laravel предлагает множество готовых компонентов и инструментов, которые позволяют быстро начать работу над проектом, не тратя время на настройку инфраструктуры.

Vue.js — это прогрессивный фреймворк для создания пользовательских интерфейсов и одностраничных приложений (SPA). Он используется для разработки клиентской части (фронтенда) веб-приложений и позволяет создавать динамические и интерактивные сайты.

Основные особенности Vue.js:

* Компонентный подход — код разбивается на пере используемые компоненты, что упрощает разработку и поддержку.
* Реактивность — изменения данных автоматически обновляют пользовательский интерфейс.
* Гибкость — можно использовать как для отдельных элементов на странице, так и для создания сложных веб-приложений.
* Легкость в освоении — синтаксис понятен даже для начинающих разработчиков.
* Интеграция с другими технологиями — Vue можно легко использовать с Laravel, что удобно при разработке интернет-магазинов.

## Используемое программное обеспечение

Draw.io — это бесплатный онлайн инструмент для создания диаграмм и схем. Он предоставляет широкий набор инструментов и элементов, которые позволяют пользователям создавать различные типы диаграмм, такие как организационные диаграммы, блок-схемы, ER-диаграммы, сетевые диаграммы, диаграммы потока процессов и многое другое.

Open Server Panel — это бесплатное программное обеспечение для Windows, которое предоставляет локальную среду разработки для веб-приложений. Он включает в себя веб-сервер Apache, базу данных MySQL, сервер приложений PHP и другие инструменты, необходимые для разработки и тестирования веб-приложений.

PhpStorm — это интегрированная среда разработки (IDE) для языка программирования PHP, разработанная компанией JetBrains. PhpStorm предоставляет разработчикам мощные инструменты для написания кода, отладки, тестирования и поддержки веб-приложений на PHP.

Postman — это популярный инструмент для разработки, тестирования и документирования API. Он предоставляет разработчикам удобный способ отправлять HTTP-запросы к веб-сервисам и анализировать ответы.

Figma — это облачный инструмент для дизайна пользовательских интерфейсов (UI) и совместной работы дизайнеров, разработчиков и других участников команды. Он позволяет создавать макеты сайтов, мобильных приложений, интерактивные прототипы и работать над дизайном в режиме реального времени.

# ГЛАВА 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ ИНФОРМАЦИОННОГО ПРОДУКТА

## Описание проекта и задач

**ОБЩЕЕ ОПИСАНИЕ ПРОЕКТА**

В условиях стремительного развития цифровых технологий и роста популярности онлайн-торговли, создание удобной, надёжной и функциональной платформы для интернет-магазина становится важной задачей. Проект "Тайны Востока" представляет собой специализированный веб-сервис по продаже одежды из азиатских стран (Японии, Кореи, Китая), ориентированный на широкий круг пользователей.

Разрабатываемая информационная система призвана обеспечить удобный доступ к товарам, упростить процесс оформления заказов, а также обеспечить эффективную работу персонала и администрирование магазина. Основной упор сделан на создание гибкой архитектуры, построенной по принципам клиент-серверного взаимодействия с использованием REST API.

В рамках проекта предусмотрено чёткое разграничение прав и функциональных возможностей пользователей. Система разделяет пользователей на три основные группы: клиентов, менеджеров и администраторов, каждая из которых выполняет определённые функции в рамках платформы.

**Функционал неавторизированного пользователя:**

1. Авторизация.
2. Регистрация.
3. Просмотр всех товаров.
4. Просмотр конкретного товара.
5. Фильтр товаров по категории, стране, цене и полу.
6. Поиск товаров по описанию или названию.
7. Просмотр категорий.
8. Просмотр стран.
9. Просмотр отзывов у товаров.
10. Просмотр пунктов выдачи.
11. Просмотр цветов.
12. Просмотр размеров.

**Функционал клиента:**

1. Просмотр, редактирование своей корзины.
2. Добавление и удаление товаров из своей корзины.
3. Оформление, просмотр, отмена своих заказов.
4. Просмотр конкретных заказов и их статуса.
5. Просмотр, добавление и удаление своих отзывов на купленные товары.

**Функционал менеджера:**

1. Просмотр, создание (с возможностью добавить новый цвет и размер), редактирование и удаление товаров.
2. Просмотр, редактирование (изменение статуса) и отмена заказов.
3. Просмотр всех заказов.
4. Удаление отзывов клиентов.

**Функционал администратора:**

1. Просмотр, редактирование всех пользователей.
2. Просмотр конкретного пользователя.
3. Создание категорий.
4. Просмотр, регистрация и удаление менеджеров.
5. Просмотр, создание (с возможностью добавить новый цвет и размер), редактирование и удаление товаров.
6. Просмотр, редактирование (изменение статуса) и отмена заказов.
7. Добавление, редактирование и удаление пункта выдачи.
8. Удаление отзывов клиентов.

**Функционал авторизированного пользователя:**

1. Выход из системы.
2. Просмотр и редактирование своего профиля.

**ОБЩИЕ ТРЕБОВАНИЯ**

Сервер не может или не будет обрабатывать запрос из-за чего-то, что очевидно является ошибкой клиента:

|  |
| --- |
| Неверный формат запроса |
| "code": 400,  "message": "Bad Request" |

Для доступа к ресурсу требуется аутентификация, и клиент не предоставил действительные учетные данные:

|  |
| --- |
| Не авторизован |
| «code»: 401,  «message»: «Unauthorized» |

Сервер понял запрос, но отказывается его авторизовать. Это может быть связано с отсутствием необходимых прав доступа:

|  |
| --- |
| Доступ запрещен |
| «code»: 403,  «message»: «Forbidden» |

Сервер не может найти запрошенный ресурс. Это может быть связано с неправильным URL или удалением ресурса:

|  |
| --- |
| Ресурс не найден |
| «code»: 404,  «message»: «Not Found» |

В случае ошибок, связанных с валидацией данных во всех запросах необходимо возвращать следующее:

|  |
| --- |
| Тело ответа |
| “code”: <code>,  “message”: <message> |

Вместо <code> и <message> необходимо указывать соответствующее значение, определенное в описании ответа на соответствующий запрос.

## Ролевая модель

Исходя из основных требований функциональности системы можно построить таблицу, разделяя роли и функциональность (см. таблицу 1):

Таблица 1. Ролевая модель

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Раздел** | **Функционал** | **Гость** | **Клиент** | **Менеджер** | **Администратор** |
| Авторизация | Авторизация | Нет | Да | Да | Да |
| Регистрация | Да | Да | Нет | Нет |
| Выход | Нет | Да | Да | Да |
| Товары | Просмотр всех товаров | Да | Да | Да | Да |
| Создание товара | Нет | Нет | Да | Да |
| Просмотр товара | Да | Да | Да | Да |
| Обновление товара | Нет | Нет | Да | Да |
| Удаление товара | Нет | Нет | Да | Да |
| Профиль | Просмотр профиля | Нет | Да | Да | Да |
| Редактирование профиля | Нет | Да | Да | Да |
| Фильтры | По параметрам | Да | Да | Да | Да |
| По названию или описанию | Да | Да | Да | Да |
| Страны | Просмотр всех стран | Да | Да | Да | Да |
| Категории | Просмотр всех категорий | Да | Да | Да | Да |
| Создание категории | Нет | Нет | Да | Да |
| Отзывы | Просмотр всех отзывов по товару | Да | Да | Да | Да |
| Создание отзыва | Нет | Да | Нет | Нет |
| Удаление отзыва | Нет | Да | Да | Да |
| Пункты выдачи | Просмотр всех пунктов выдачи | Да | Да | Да | Да |
| Создание пункта выдачи | Нет | Нет | Нет | Да |
| Обновление пункта выдачи | Нет | Нет | Нет | Да |
| Удаление пункта выдачи | Нет | Нет | Нет | Да |
| Корзина | Просмотр корзины | Нет | Да | Нет | Нет |
| Добавление товара в корзину | Нет | Да | Нет | Нет |
| Обновление количества товара | Нет | Да | Нет | Нет |
| Удаление товара из корзины | Нет | Да | Нет | Нет |
| Заказы | Просмотр своих заказов | Нет | Да | Нет | Нет |
| Просмотр всех заказов | Нет | Нет | Да | Да |
| Создание заказа | Нет | Да | Нет | Нет |
| Просмотр своего заказа | Нет | Да | Нет | Нет |
| Просмотр чужого заказа | Нет | Нет | Да | Да |
| Обновление заказа | Нет | Нет | Да | Да |
| Отмена заказа | Нет | Да | Да | Да |
| Менеджеры | Просмотр всех менеджеров | Нет | Нет | Нет | Да |
| Создание менеджера | Нет | Нет | Нет | Да |
| Удаление менеджера | Нет | Нет | Нет | Да |
| Пользователи | Просмотр всех пользователей | Нет | Нет | Нет | Да |
| Просмотр пользователя | Нет | Нет | Нет | Да |
| Обновление пользователя | Нет | Нет | Нет | Да |
| Статусы | Просмотр всех статусов | Да | Да | Да | Да |

## Специфические требования к API

**Авторизация**

Запрос для аутентификации пользователя в системе. При отправке запроса необходимо передать объект с почтой и паролем. Если пользователь отправил корректные данные, то необходимо вернуть сгенерированный токен, а иначе сообщение об ошибке.

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/login |
| Заголовки |
| Content-Type: application/json  Accept: application/json |
| Тело запроса |
| {     "password": "Password123@",     "email": "abdul@mail.ru"  } |
| Ответ на успешную авторизацию (200 OK) |
| {      "token": "$2y$12$22NQ9pW9FwbHX3tTWmShpurXq34vH7FGpogPBzQEykHYtE3fhREHq",      "user": {          "id": 9,          "name": "Абдул Абдурихманович",          "email": "abdul@mail.ru",          "telephone": "88005553509",          "sex": 1,          "role\_id": 3,          "created\_at": "2025-05-15T03:31:55.000000Z",          "updated\_at": "2025-06-11T06:08:34.000000Z"      }  } |
| Ошибка авторизации (401 Unauthorized) |
| {      "code": 401,      "message": "Неверная почта или пароль"  } |

**Регистрация**

Запрос для регистрации клиента в системе. При отправке запроса необходимо передать объект с именем, полом, номером телефона, почтой и паролем. Если пользователь отправил корректные данные, то необходимо вернуть сгенерированный токен, а иначе сообщение об ошибке.

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/register |
| Заголовки |
| Authorization: application/json  Accept: application/json |
| Тело запроса |
| {     "name": "Абдул Абдурихманови",     "email": "abdulchikп@mail.ru",     "telephone": "88005553535",     "sex": 1,     "password": "Lexa2019@"  } |
| Ответ на успешную регистрацию (201 OK) |
| {      "user": {          "name": "Абдул Абдурихманови",          "email": "abdulchikп@mail.ru",          "telephone": "88005553535",          "sex": 1,          "role\_id": 3,          "updated\_at": "2025-06-11T06:12:57.000000Z",          "created\_at": "2025-06-11T06:12:57.000000Z",          "id": 14      },      "token": "$2y$12$MSVG5IKxWXWgqxSowxP55OGPUfAWFMkgNuDhhX.ZWAFNeG4dWFwNi"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {      "code": 422,      "message": "Ошибка валидации данных"  } |

**Выход**

Запрос предназначен для очистки значение токена пользователя.

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/logout |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Тело запроса |
| {  token: $2y$12$hgKj3HcOQDcVzbfZhRILq/ZGXVUAW2mYn5meSUC  } |
| Ответ на успешный выход из системы (200 OK) |
| {      "message": "Вы вышли из системы"  } |

**CRUD-операции по товару**

Get-запрос к /api/product с токеном в теле запроса возвращает список товаров (200 ok) или сообщение о том, что товары не найдены (404 not found).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/product |
| Заголовки |
| Authorization: application/json  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| [      {          "id": 1,          "photo": "http://secrets-of-the-east.ru/storage/products/girl8.jpg",          "name": "Футболки и топы 1",          "description": "Описание товара",          "sex": 1,          "quantity": 6,          "price": "4302.00",          "category\_id": 3,          "country\_id": 1,          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 2,          "photo": "http://secrets-of-the-east.ru/storage/products/girl5.jpg",          "name": "Свитшоты и худи 2",          "description": "Описание товара",          "sex": 1,          "quantity": 5,          "price": "1655.00",          "category\_id": 2,          "country\_id": 3,          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      }  ] |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Не найдено"  } |

Post-запрос к /api/product с описанием товара в теле и токеном в теле запроса создаёт новый товар (200 ok), либо доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/product |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "name": "Футболка Premium",      "description": "Качественная хлопковая футболка",      "sex": 1,      "price": 1999,      "category\_id": 4,      "country\_id": 1,      "colors": [          {              "color\_id": 3,              "sizes": [                  {                      "size\_id": 1,                      "quantity": 10                  },                  {                      "size\_id": 2,                      "quantity": 15                  }              ]          },          {              "new\_color\_name": "Мятный",              "new\_color\_hex": "#A2F2D5",              "sizes": [                  {                      "new\_size\_name": "XL",                      "quantity": 8                  },                  {                      "size\_id": 2,                      "quantity": 12                  }              ]          }      ]  } |
| Ответ на успешное создание (200 OK) |
| {      "product": {          "photo": **null**,          "name": "Футболка Premium",          "description": "Качественная хлопковая футболка",          "sex": 1,          "quantity": 45,          "price": 1999,          "category\_id": 4,          "country\_id": 1,          "updated\_at": "2025-06-11T06:34:31.000000Z",          "created\_at": "2025-06-11T06:34:31.000000Z",          "id": 78      },      "message": "Товар успешно создан с цветами и размерами"  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "У вас нет прав на выполнение этого действия"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {      "code": 422,      "message": "Ошибка валидации данных"  } |

Patch-запрос к /api/product/{id} с обновленным описанием товара и токеном в теле запроса обновляет существующий товар (200 ok) или доступа (403 forbidden), сообщение о том, что товар не найден (404 not found) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/product/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "name": "Футболка Premium",      "description": "Качественная хлопковая футболка",      "sex": 1,      "price": 1999,      "category\_id": 4,      "country\_id": 1,      "colors": [          {              "color\_id": 3,              "sizes": [                  {                      "size\_id": 1,                      "quantity": 10                  },                  {                      "size\_id": 2,                      "quantity": 15                  }              ]          },          {              "new\_color\_name": "Мятный",              "new\_color\_hex": "#A2F2D5",              "sizes": [                  {                      "new\_size\_name": "XL",                      "quantity": 8                  },                  {                      "size\_id": 2,                      "quantity": 12                  }              ]          }      ]  } |
| Ответ на успешное обновление (200 OK) |
| {      "product": {          "id": 9,          "photo": "http://secrets-of-the-east.ru/storage/products/girl2.jpg",          "name": "Футболка Premium",          "description": "Качественная хлопковая футболка",          "sex": 1,          "quantity": 45,          "price": 1999,          "category\_id": 4,          "country\_id": 1,          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-06-11T06:37:56.000000Z",          "product\_color\_sizes": [              {                  "id": 582,                  "quantity": 10,                  "product\_id": 9,                  "color\_id": 3,                  "size\_id": 1,                  "created\_at": "2025-06-11T06:37:56.000000Z",                  "updated\_at": "2025-06-11T06:37:56.000000Z",                  "color": {                      "id": 3,                      "name": "Красный",                      "hex": "#FF0000",                      "created\_at": "2025-05-15T03:31:56.000000Z",                      "updated\_at": "2025-05-15T03:31:56.000000Z"                  },                  "size": {                      "id": 1,                      "name": "26-27",                      "created\_at": "2025-05-15T03:31:56.000000Z",                      "updated\_at": "2025-05-15T03:31:56.000000Z"                  }  }          ]      },      "message": "Товар успешно обновлен"  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "У вас нет прав на выполнение этого действия"  } |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Не найдено"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {      "code": 422,      "message": "Ошибка валидации данных"  } |

Delete-запрос к /api/product/{id} с токеном в теле запроса удаляет товар (200 ok) или доступа (403 forbidden) или сообщение о том, что товар не найден (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: delete  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/product/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешное удаление (200 OK) |
| {      "message": "Товар успешно удален",      "success": **true**  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "У вас нет прав на выполнение этого действия"  } |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Не найдено"  } |

Get-запрос к /api/product/{id} с токеном в теле запроса возвращает товар по его id (200 ok) или сообщение о том, что товар не найден (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/product/{id} |
| Заголовки |
| Authorization: application/json  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| {      "id": 50,      "photo": "http://secrets-of-the-east.ru/storage/products/girl2.jpg",      "name": "Рубашки и блузы 50",      "description": "Описание товара",      "sex": 0,      "quantity": 9,      "price": "3602.00",      "category\_id": 4,      "country\_id": 2,      "created\_at": "2025-05-15T03:31:56.000000Z",      "updated\_at": "2025-05-15T03:31:56.000000Z",      "category": {          "id": 4,          "name": "Рубашки и блузы",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      "country": {          "id": 2,          "name": "Китай",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      "product\_color\_sizes": [          {              "id": 410,              "quantity": 6,              "product\_id": 50,              "color\_id": 6,              "size\_id": 6,              "created\_at": "2025-05-15T03:31:56.000000Z",              "updated\_at": "2025-05-15T03:31:56.000000Z",              "color": {                  "id": 6,                  "name": "Жёлтый",                  "hex": "#FFFF00",                  "created\_at": "2025-05-15T03:31:56.000000Z",                  "updated\_at": "2025-05-15T03:31:56.000000Z"              },              "size": {                  "id": 6,                  "name": "36-37",                  "created\_at": "2025-05-15T03:31:56.000000Z",                  "updated\_at": "2025-05-15T03:31:56.000000Z"              }          }      ]  } |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Не найдено"  } |

**CRUD-операции по профилю**

Get-запрос к /api/profile с токеном в теле запроса возвращает информацию об авторизированном пользователе (200 ok).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/profile |
| Заголовки |
| Authorization: application/json  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| {      "id": 10,      "name": "Алексей Петрович",      "email": "ruginpetr@mail.ru",      "telephone": "88005553510",      "sex": 1,      "role": "Пользователь",      "created\_at": "2025-05-15T03:31:55.000000Z",      "updated\_at": "2025-05-23T09:10:47.000000Z"  } |

Patch-запрос к /api/profile с описанием профилем авторизированного пользователя и токеном в теле запроса обновляет существующего пользователя (200 ok) или ошибку валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/profile |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "name": "Лёха"  } |
| Ответ на успешное обновление (200 OK) |
| {      "message": "Профиль успешно обновлен",      "data": {          "id": 10,          "name": "Лёха",          "email": "ruginpetr@mail.ru",          "telephone": "88005553510",          "sex": 1,          "role\_id": 3,          "created\_at": "2025-05-15T03:31:55.000000Z",          "updated\_at": "2025-06-11T07:00:30.000000Z",          "role": {              "id": 3,              "name": "Пользователь",              "code": "user",              "created\_at": "2025-05-15T03:31:53.000000Z",              "updated\_at": "2025-05-15T03:31:53.000000Z"          }      }  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {      "code": 422,      "message": "Ошибка валидации данных"  } |

**Фильтрация**

Get-запрос к /api/products/filter возвращает товары по заданным параметрам (200 ok) или сообщение о том, что товары по заданным фильтрам не найдены (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/products/filter?country\_id=1&category\_id=1&min\_price=1000&max\_price=4000&sex=1 |
| Заголовки |
| Authorization: application/json  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| {      "success": **true**,      "filters": {          "category\_id": "1",          "country\_id": "1",          "min\_price": "1000",          "max\_price": "4000",          "sex": "1"      },      "count": 1,      "data": [          {              "id": 55,              "photo": "http://secrets-of-the-east.ru/storage/products/girl5.jpg",              "name": "Верхняя одежда 55",              "description": "Описание товара",              "sex": 1,              "quantity": 5,              "price": "3549.00",              "category\_id": 1,              "country\_id": 1,              "created\_at": "2025-05-15T03:31:56.000000Z",              "updated\_at": "2025-05-15T03:31:56.000000Z",              "category": {                  "id": 1,                  "name": "Верхняя одежда",                  "created\_at": "2025-05-15T03:31:56.000000Z",                  "updated\_at": "2025-05-15T03:31:56.000000Z"              },              "country": {                  "id": 1,                  "name": "Корея",                  "created\_at": "2025-05-15T03:31:56.000000Z",                  "updated\_at": "2025-05-15T03:31:56.000000Z"              }          }      ]  } |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Товары по заданным фильтрам не найдены"  } |

Get-запрос к /api/products/search с названием или описанием возвращает список товаров (200 ok) или сообщение о том, что товары не найдены (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/products/search |
| Заголовки |
| Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "q": "Футболки и топы 18"  } |
| Ответ на успешный просмотр (200 OK) |
| {      "success": **true**,      "query": "Футболки и топы 18",      "count": 1,      "data": [          {              "id": 18,              "photo": "http://secrets-of-the-east.ru/storage/products/girl1.jpg",              "name": "Футболки и топы 18",              "description": "Описание товара",              "sex": 0,              "quantity": 8,              "price": "3543.00",              "category\_id": 3,              "country\_id": 3,              "created\_at": "2025-05-15T03:31:56.000000Z",              "updated\_at": "2025-05-15T03:31:56.000000Z",              "category": {                  "id": 3,                  "name": "Футболки и топы",                  "created\_at": "2025-05-15T03:31:56.000000Z",                  "updated\_at": "2025-05-15T03:31:56.000000Z"              },              "country": {                  "id": 3,                  "name": "Япония",                  "created\_at": "2025-05-15T03:31:56.000000Z",                  "updated\_at": "2025-05-15T03:31:56.000000Z"              }          }      ]  } |
| Ошибка поиска (404 Not Found) |
| {      "message": "Товары по вашему запросу не найдены"  } |

**CRUD-операции по стране**

Get-запрос к /api/country с токеном в теле запроса возвращает список стран (200 ok) или сообщение о том, что страны не найдены (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/country |
| Заголовки |
| Accept: application/json  Content-Type: application/json |
| Ответ на успешный просмотр (200 OK) |
| {  “id”: 1,  “name”: “Китай”,  “created\_at”: “2025-02-26T10:15:30.000000Z”,  “updated\_at”: “2025-02-26T10:15:30.000000Z”  },  {  “id”: 2,  “name”: “Япония”,  “created\_at”: “2025-02-26T10:15:30.000000Z”,  “updated\_at”: “2025-02-26T10:15:30.000000Z”  },  {  “id”: 3,  “name”: “Корея”,  “created\_at”: “2025-02-26T10:15:30.000000Z”,  “updated\_at”: “2025-02-26T10:15:30.000000Z”  } |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Не найдено"  } |

**CRUD-операции по категории**

Get-запрос к /api/category с токеном в теле запроса возвращает список категорий одежды (200 ok) или сообщение о том, что категории одежды не найдены (404 not found).

|  |
| --- |
| Метод: get  Url: [http://secrets-of-the-east.ru/api/category](http://secrets-of-the-east.ru/api/categoryff) |
| Заголовки |
| Authorization: application/json  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| [      {          "id": 1,          "name": "Верхняя одежда",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 2,          "name": "Свитшоты и худи",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 3,          "name": "Футболки и топы",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 4,          "name": "Рубашки и блузы",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 5,          "name": "Джинсы и брюки",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 6,          "name": "Спортивная одежда",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 7,          "name": "Топчик",          "created\_at": "2025-06-04T09:47:21.000000Z",          "updated\_at": "2025-06-04T09:47:21.000000Z"      },      {          "id": 8,          "name": "Топчик",          "created\_at": "2025-06-04T12:02:13.000000Z",          "updated\_at": "2025-06-04T12:02:13.000000Z"      }  ] |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Не найдено"  } |

Post-запрос к /api/category с названием категории одежды в теле и токеном в теле запроса создаёт новую категорию одежды (200 ok), либо возвращает ошибку доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/category |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "name": "Новая категория"  } |
| Ответ на успешное создание (200 OK) |
| {      "name": "Новая категория",      "updated\_at": "2025-06-11T07:39:47.000000Z",      "created\_at": "2025-06-11T07:39:47.000000Z",      "id": 9  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "У вас нет прав на выполнение этого действия"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {      "code": 422,      "message": "Ошибка валидации данных"  } |

**CRUD-операции по отзыву**

Get-запрос к /api/product/{id}/review с токеном в теле запроса возвращает список отзывов по id товара (200 ok) или сообщение о том, что отзывы у выбранного товара не найдены (404 not found) или о том, что сам товар не найден (404 not found).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/product/{id}/review |
| Заголовки |
| Authorization: application/json  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| [      {          "id": 11,          "rating": 4,          "description": "хихиххи",          "user\_id": 1,          "product\_id": 1,          "created\_at": "2025-05-15T11:44:14.000000Z",          "updated\_at": "2025-05-15T11:44:14.000000Z",          "user": {              "id": 1,              "name": "Дурачок"          }      },      {          "id": 1,          "rating": 5,          "description": "Идеальная куртка! Смотрится дороже своей цены, все швы аккуратные. Носила всю зиму — не промокает.",          "user\_id": 3,          "product\_id": 1,          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z",          "user": {              "id": 3,              "name": "Иванов Иван Иванович"          }      }  ] |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Отзывы для данного товара не указаны"  } |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Товар не найден"  } |

Post-запрос к /api/product/{id}/review с описанием отзыва в теле и токеном в теле запроса создаёт новый отзыв (200 ok), либо ошибку плохого запроса (400 bad request), либо возвращает ошибку доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/product/{id}/review |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "rating": 5,      "description": "Хорошая вещь"  } |
| Ответ на успешное создание (201 Created) |
| {      "rating": 5,      "description": "Хорошая вещь",      "user\_id": 10,      "product\_id": "60",      "updated\_at": "2025-06-11T07:59:31.000000Z",      "created\_at": "2025-06-11T07:59:31.000000Z",      "id": 12  } |
| Ошибка доступа (400 Bad Request) |
| {      "message": "Вы уже оставили отзыв на этот товар"  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "Вы можете оставить отзыв только после покупки и завершения заказа"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {      "code": 422,      "message": "Ошибка валидации данных"  } |

Delete-запрос к /api/product/{productId}/review/{reviewId} с токеном в теле запроса удаляет категорию одежды (200 ok) или возвращает ошибку плохого запроса (400 bad request) или возвращает ошибку доступа (403 forbidden) или сообщение о том, что отзыв не найден (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: delete  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/product/{productId}/review/{reviewId} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешное удаление (200 OK) |
| {      "message": "Отзыв успешно удален"  } |
| Ошибка авторизации (400 Bad Request) |
| {      "code": 400,      "message": "Этот отзыв не принадлежит указанному товару"  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "Недостаточно прав для удаления отзыва"  } |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Отзыв не найден"  } |

**CRUD-операции по пунктам выдачи**

Get-запрос к /api/point возвращает список пунктов выдачи (200 ok) или сообщение о том, что пунктов выдачи нет (404 not found).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/point |
| Заголовки |
| Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| [      {          "id": 1,          "city": "Томск",          "street": "Иркутский тракт",          "house": "175",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 2,          "city": "Томск",          "street": "Иркутский тракт",          "house": "173",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 3,          "city": "Москва",          "street": "Тверская",          "house": "10",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      }  ] |
| Ошибка поиска (404 Not Found) |
| {      "code": 404,      "message": "Не найдено"  } |

Post-запрос к /api/point с описанием адреса в теле запроса создаёт новый пункт выдачи (200 ok), либо ошибку плохого запроса (400 bad request), либо возвращает ошибку доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/point |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "city": "Город",      "street": "Улица",      "house": "Дом"  } |
| Ответ на успешное создание (201 Created) |
| {      "city": "Город",      "street": "Улица",      "house": "Дом",      "updated\_at": "2025-06-11T08:26:41.000000Z",      "created\_at": "2025-06-11T08:26:41.000000Z",      "id": 12  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "У вас нет прав на выполнение этого действия"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {      "code": 422,      "message": "Ошибка валидации данных"  } |

Patch-запрос к /api/point/{id} с описанием пункта выдачи авторизированного пользователя и токеном в теле запроса обновляет существующий пункт выдачи (200 ok) или возвращает ошибку доступа (403 forbidden).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/point/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "city": "Томск"  } |
| Ответ на успешное обновление (200 OK) |
| {      "id": 12,      "city": "Томск",      "street": "Улица",      "house": "Дом",      "created\_at": "2025-06-11T08:26:41.000000Z",      "updated\_at": "2025-06-11T08:30:40.000000Z"  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "У вас нет прав на выполнение этого действия"  } |
| Ошибка валидации (422 Unprocessable Entity) |
| {      "code": 422,      "message": "Ошибка валидации данных"  } |

Delete-запрос к /api/point/{id} авторизированного пользователя и токеном в теле запроса удаляет существующий пункт выдачи (200 ok) или возвращает ошибку доступа (403 forbidden) или возвращает ошибку валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: delete  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/point/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Ответ на успешное удаление (200 OK) |
| {      "message": "Пункт выдачи удалён."  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "У вас нет прав на выполнение этого действия"  } |

**CRUD-операции по корзине**

Get-запрос к /api/cart с токеном в теле запроса возвращает список товаров корзины текущего пользователя (200 ok) или сообщение о том, что корзина пуста (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/cart |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| {      "total\_cost": 23585,      "created\_at": "2025-06-11T08:40:18.000000Z",      "updated\_at": "2025-06-11T08:40:18.000000Z",      "products": [          {              "id": 501,              "name": "Спортивная одежда 60",              "price": "4717.00",              "photo": "http://secrets-of-the-east.ru/storage/products/girl6.jpg",              "description": "Описание товара",              "color": "Белый",              "size": "30-31",              "quantity": 5          }      ]  } |
| Ошибка поиска (404 Not Found) |
| {      "message": "Корзина пуста"  } |

Post-запрос к /api/cart/product/{productColorSizeId} с указанием количества в теле запроса добавляет товар в корзину (200 ok), либо ошибку плохого запроса (400 bad request), либо возвращает ошибку доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/cart/product/{productColorSizeId} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "quantity": 5  } |
| Ответ на успешное добавление (200 OK) |
| {      "status": "success",      "message": "Товар добавлен в корзину"  } |
| Ошибка поиска (404 Not Found) |
| {      "message": "Товар не найден"  } |

Patch-запрос к /api/cart/product/{productColorSizeId} с указанием количества и токеном в теле запроса обновляет количество товара в корзине (200 ok) или возвращает ошибку поиска (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/cart/product/{productColorSizeId} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "quantity": 5  } |
| Ответ на успешное обновление (200 OK) |
| {      "status": "success",      "message": "Количество товара в корзине успешно обновлено",      "data": {          "total\_cost": 36887      }  } |
| Ошибка поиска (404 Not Found) |
| {      "status": "error",      "message": "Товар не найден в корзине"  } |

Delete-запрос к /api/cart/product/{productColorSizeId} авторизированного пользователя и токеном в теле запроса удаляет товар из корзины (200 ok) или возвращает ошибку поиска (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: delete  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/cart/product/{productColorSize} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Ответ на успешное удаление (200 OK) |
| {      "status": "success",      "message": "Товар успешно удалён из корзины",      "data": {          "total\_cost": 32170,          "remaining\_items": 1      }  } |
| Ошибка поиска (404 Not Found) |
| {      "status": "error",      "message": "Указанный товар не найден в вашей корзине"  } |

**CRUD-операции по заказам**

Get-запрос к /api/order с токеном в теле запроса возвращает список заказов текущего пользователя или список всех заказов (в зависимости, кто выполнил запрос) (200 ok) или сообщение о том, что заказы отсутсвуют (404 not found).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/order |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| {      "status": "success",      "data": [          {              "id": 11,              "order\_date": "2025-06-04 11:57:30",              "status": "Отменён",              "total\_cost": 47170,              "user": "Дурачок",              "address": {                  "city": "Томск",                  "street": "Иркутский тракт",                  "house": "175"              },              "items\_count": 2          },          {              "id": 12,              "order\_date": "2025-06-11 07:57:15",              "status": "Доставлен",              "total\_cost": 23585,              "user": "Лёха",              "address": {                  "city": "Томск",                  "street": "Иркутский тракт",                  "house": "175"              },              "items\_count": 1          }      ]  } |
| Ошибка поиска (для клиента) (404 Not Found) |
| {      "message": "У вас пока нет заказов"  } |
| Ошибка поиска (для менеджеров и администраторов) (404 Not Found) |
| {      "message": "В системе пока нет заказов"  } |

Post-запрос к /api/order с указанием пункта выдачи в теле запроса возвращает ссылку на сайт для тестовой оплаты (200 ok), либо ошибку плохого запроса (400 bad request), либо возвращает ошибку доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/order |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {    "point\_id": 1  } |
| Ответ на успех (200 OK) |
| {      "status": "payment\_required",      "message": "Требуется подтверждение оплаты",      "payment\_url": "https://checkout.stripe.com/c/pay/cs\_test\_a1hQXany8puo5NJq3XRTnzQjwrbvHTHuKNIzKU5lLC0haLBxYY73rtKBff#fidkdWxOYHwnPyd1blpxYHZxWjA0V1U1ajRDVWJvMENWM0JMdkZxf2tJbHZNVU9kMDNfSn1wMW10XzwySVxgfGJQY3NyVzNidWlIU1Fta2BvTzZOV0ZnUlZIM19ib3F%2FS258TldIcDJiU3BsNTVpTXVpdm9AaycpJ2N3amhWYHdzYHcnP3F3cGApJ2lkfGpwcVF8dWAnPyd2bGtiaWBabHFgaCcpJ2BrZGdpYFVpZGZgbWppYWB3dic%2FcXdwYHgl",      "session\_id": "cs\_test\_a1hQXany8puo5NJq3XRTnzQjwrbvHTHuKNIzKU5lLC0haLBxYY73rtKBff"  } |
| Ошибка плохого запроса (400 Bad Request) |
| {      "status": "error",      "message": "Не указан адрес доставки."  } |
| Ошибка плохого запроса (400 Bad Request) |
| {      "status": "error",      "message": "Корзина пуста. Добавьте товары для оформления заказа."  } |

Patch-запрос к /api/order/{id} с указанием нового статуса и токеном в теле запроса обновляет статус заказа (200 ok) или возвращает ошибку поиска (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/order/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {      "status\_id": 1  } |
| Ответ на успешное обновление (200 OK) |
| {      "message": "Заказ обновлён",      "order": {          "id": 12,          "order\_date": "2025-06-11 07:57:15",          "total": "23585.00",          "user\_id": 10,          "status\_id": 1,          "point\_id": 1,          "created\_at": "2025-06-11T07:57:15.000000Z",          "updated\_at": "2025-06-11T09:19:31.000000Z"      }  } |
| Ошибка поиска (404 Not Found) |
| {      "error": "Заказ не найден"  } |

Patch-запрос к /api/order/cancelled/{id} с токеном в теле запроса обновляет статус заказа на «Отменён» (200 ok) или возвращает ошибку поиска (404 not found).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/order/cancelled/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Ответ на успешное обновление (200 OK) |
| {      "message": "Заказ отменён"  } |
| Ошибка поиска (404 Not Found) |
| {      "error": "Заказ не найден"  } |

Get-запрос к /api/order/{id} с токеном в теле запроса возвращает информацию о заказе (в зависимости, кто выполнил запрос) (200 ok) или сообщение о том, что заказ не найден (404 not found).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/order/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| {      "id": 12,      "order\_date": "2025-06-11 07:57:15",      "status": "Отменён",      "total\_cost": 23585,      "user": "Лёха",      "address": {          "city": "Томск",          "street": "Иркутский тракт",          "house": "175"      },      "items": [          {              "product\_id": 60,              "photo": "http://secrets-of-the-east.ru/storage/products/girl6.jpg",              "name": "Спортивная одежда 60",              "price": "4717.00",              "quantity": 5,              "color": "Белый",              "size": "30-31",              "total": 23585          }      ]  } |
| Ошибка поиска (404 Not Found) |
| {      "error": "Заказ не найден"  } |

**CRUD-операции по менеджерам**

Get-запрос к /api/manager с токеном в теле запроса возвращает список менеджеров (200 ok) или сообщение о том, что у пользователя нет доступа (403 forbidden).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/manager |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| [      {          "id": 2,          "name": "Менеджер Отдела",          "email": "manager@mail.ru",          "telephone": "88005553502",          "role": "Менеджер",          "created\_at": "2025-05-15 03:31:53"      },      {          "id": 13,          "name": "manager",          "email": "manage@mail.ru",          "telephone": "88888888888",          "role": "Менеджер",          "created\_at": "2025-05-29 09:05:02"      }  ] |
| Ошибка доступа (403 Forbidden) |
| {      "message": "Доступ запрещен"  } |

Post-запрос к /api/manager с указанием информации о новом менеджере в теле запроса возвращает успешный ответ (200 ok), либо возвращает ошибку доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: post  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/manager |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {     "name": "Абдул Абдурихманови",     "email": "abdulchi@mail.ru",     "telephone": "88005553535",     "sex": 1,     "password": "Lexa2019@"  } |
| Ответ на успешное создание (201 Created) |
| {      "user": {          "name": "Абдул Абдурихманови",          "email": "abdulchi@mail.ru",          "telephone": "88005553535",          "sex": 1,          "role\_id": 2,          "updated\_at": "2025-06-11T09:40:14.000000Z",          "created\_at": "2025-06-11T09:40:14.000000Z",          "id": 15      },      "token": "$2y$12$cWz0fEZVYZTuLAXe.Ib16OvSV5/3ZXLhq8E20PS/o8FXhRHZo1hy2",      "message": "Менеджер успешно создан"  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "У вас нет прав на выполнение этого действия"  } |
| Ошибка плохого запроса (422 Unprocessable Entity) |
| {      "code": 422,      "message": "Ошибка валидации данных"  } |

Delete-запрос к /api/manager/{id} авторизированного пользователя и токеном в теле запроса удаляет товар из корзины (200 ok) или возвращает ошибку плохого запроса (400 bad request) или возвращает ошибку доступа (403 forbidden).

|  |
| --- |
| HTTP-метод и URL |
| Метод: delete  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/manager/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Ответ на успешное удаление (200 OK) |
| {      "message": "Менеджер успешно удален",      "deleted\_manager": {          "id": 15,          "name": "Абдул Абдурихманови",          "email": "abdulchi@mail.ru",          "telephone": "88005553535",          "role": "Менеджер",          "created\_at": "2025-06-11 09:40:14"      }  } |
| Ошибка плохого запроса (400 Bad Request) |
| {      "message": "Можно удалять только менеджеров"  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "Недостаточно прав для удаления менеджера"  } |

**CRUD-операции по клиентам**

Get-запрос к /api/user с токеном в теле запроса возвращает список пользователей (200 ok) или сообщение о том, что у пользователя нет доступа (403 forbidden).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/user |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| [      {          "id": 1,          "name": "Дурачок",          "email": "admin@mail.ru",          "telephone": "88005553501",          "sex": 1,          "role": {              "id": 1,              "name": "Администратор",              "code": "admin"          },          "created\_at": "2025-05-15T03:31:53.000000Z",          "updated\_at": "2025-06-11T06:18:30.000000Z"      },      {          "id": 2,          "name": "Менеджер Отдела",          "email": "manager@mail.ru",          "telephone": "88005553502",          "sex": 1,          "role": {              "id": 2,              "name": "Менеджер",              "code": "manager"          },          "created\_at": "2025-05-15T03:31:53.000000Z",          "updated\_at": "2025-05-15T03:31:53.000000Z"      },  …  ] |
| Ошибка доступа (403 Forbidden) |
| {      "message": "Доступ запрещен"  } |

Get-запрос к /api/user/{id} с токеном в теле запроса возвращает пользователя (200 ok) или сообщение о том, что у пользователя нет доступа (403 forbidden).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/user/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| {      "id": 1,      "name": "Дурачок",      "email": "admin@mail.ru",      "telephone": "88005553501",      "sex": 1,      "role": {          "id": 1,          "name": "Администратор",          "code": "admin"      },      "created\_at": "2025-05-15T03:31:53.000000Z",      "updated\_at": "2025-06-11T06:18:30.000000Z"  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "Доступ запрещен"  } |

Patch-запрос к /api/user/{id} с указанием новой информации о пользователе в теле запроса возвращает успешный ответ (200 ok), либо возвращает ошибку доступа (403 forbidden) или валидации (422 unprocessable entity).

|  |
| --- |
| HTTP-метод и URL |
| Метод: patch  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/user/{id} |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json  Content-Type: application/json |
| Тело запроса |
| {     "name": "Абдул Абдурихманови",     "email": "abdulchi@mail.ru",     "telephone": "88005553535",     "sex": 1,     "password": "Lexa2019@"  } |
| Ответ на успешное создание (201 Created) |
| {      "user": {          "id": 1,          "name": "Абдул Абдурихманови",          "email": "abdulcfefefhi@mail.ru",          "telephone": "88005553535",          "sex": 1,          "role": {              "id": 1,              "name": "Администратор",              "code": "admin"          },          "created\_at": "2025-05-15T03:31:53.000000Z",          "updated\_at": "2025-06-11T11:10:08.000000Z"      },      "message": "Данные пользователя обновлены"  } |
| Ошибка доступа (403 Forbidden) |
| {      "message": "Доступ запрещен"  } |
| Ошибка плохого запроса (422 Unprocessable Entity) |
| {      "code": 422,      "message": "Ошибка валидации данных"  } |

**Вывод цветов, размеров и статусов**

Get-запрос к /api/colors с токеном в теле запроса возвращает список цветов (200 ok) или сообщение о том, что у пользователя нет доступа (403 forbidden).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/colors |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| [      {          "id": 1,          "name": "Чёрный",          "hex": "#000000",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 2,          "name": "Белый",          "hex": "#FFFFFF",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },  …  ] |
| Ошибка доступа (404 Not Found) |
| {      "code": 404,      "message": "Не найдено"  } |

Get-запрос к /api/sizes с токеном в теле запроса возвращает список размеров (200 ok) или сообщение о том, что у пользователя нет доступа (403 forbidden).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/sizes |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| [      {          "id": 1,          "name": "26-27",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 2,          "name": "28-29",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },  …  ] |
| Ошибка доступа (404 Not Found) |
| {      "code": 404,      "message": "Не найдено"  } |

Get-запрос к /api/statuses с токеном в теле запроса возвращает список статусов (200 ok) или сообщение о том, что у пользователя нет доступа (403 forbidden).

|  |
| --- |
| Метод: get  Url: [http://](http://courseproject)secrets-of-the-east.ru/api/statuses |
| Заголовки |
| Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9  Accept: application/json |
| Ответ на успешный просмотр (200 OK) |
| [      {          "id": 1,          "name": "Принят",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },      {          "id": 2,          "name": "В сборке",          "created\_at": "2025-05-15T03:31:56.000000Z",          "updated\_at": "2025-05-15T03:31:56.000000Z"      },  …  ] |
| Ошибка доступа (404 Not Found) |
| {      "code": 404,      "message": "Не найдено"  } |

## Разработка прототипа пользовательского интерфейса

Прототип пользовательского интерфейса интернет-магазина «Тайны Востока» разработан для визуализации структуры и функционала будущего веб-сайта. Он включает главную страницу с каталогом товаров, систему фильтрации по категориям, страницу товара с подробным описанием, а также модули оформления заказа и личного кабинета. Макет ориентирован на удобство навигации, интуитивный дизайн и адаптивность для разных устройств.

**Главная страница**

При входе на сайт пользователь попадает на главную страницу, где сразу отображаются товары. Дизайн ориентирован на удобство навигации и быстрый доступ к нужным категориям.

**Верхняя панель навигации**

В верхней части сайта расположена **основная навигационная панель**, включающая:

**Середина**:

* **«Пункт выдачи»** – выбор удобного места получения заказа.
* **«Профиль»** – личный кабинет пользователя.
* **«Корзина»** – просмотр добавленных товаров и оформление заказа.
* **«Заказы» -** просмотр своих заказов и отмена их по надобности.
* **Кнопка «Инструменты»** – отображается только у **администраторов и менеджеров**. При нажатии открывается **выпадающее меню** с возможностью управления контентом сайта.

**Функционал кнопки «Инструменты»**

**Менеджеры могут:**

* Управлять товарами (создавать, редактировать, удалять).
* Редактировать статусы заказов (подтверждение, отмена и тд).
* Просматривать все заказы.
* Управлять отзывами (просмотр и удаление).

**Администраторы могут:**

* Обладать всем функционалом менеджеров.
* Управлять пользователями (просмотр, редактирование, регистрация и удаление менеджеров).
* Управлять пунктами выдачи.
* Управлять категориями одежды.

**Фильтрация**

Фильтрация товаров расположена **в зоне товаров**, которая находится слева:

**Слева от товаров** – категории по **типу одежды (**худи, толстовки, свитшоты, футболки и т. д**)**, **страна, пол и фильтр по цене.**

**Поиск товаров**

В верхней части страницы предусмотрена поисковая строка, позволяющая пользователю быстро находить товары по названию, бренду или характеристикам.

Таким образом, главная страница объединяет удобную навигацию, фильтрацию товаров и инструменты управления. Все элементы интерфейса представлены на рисунке 1.

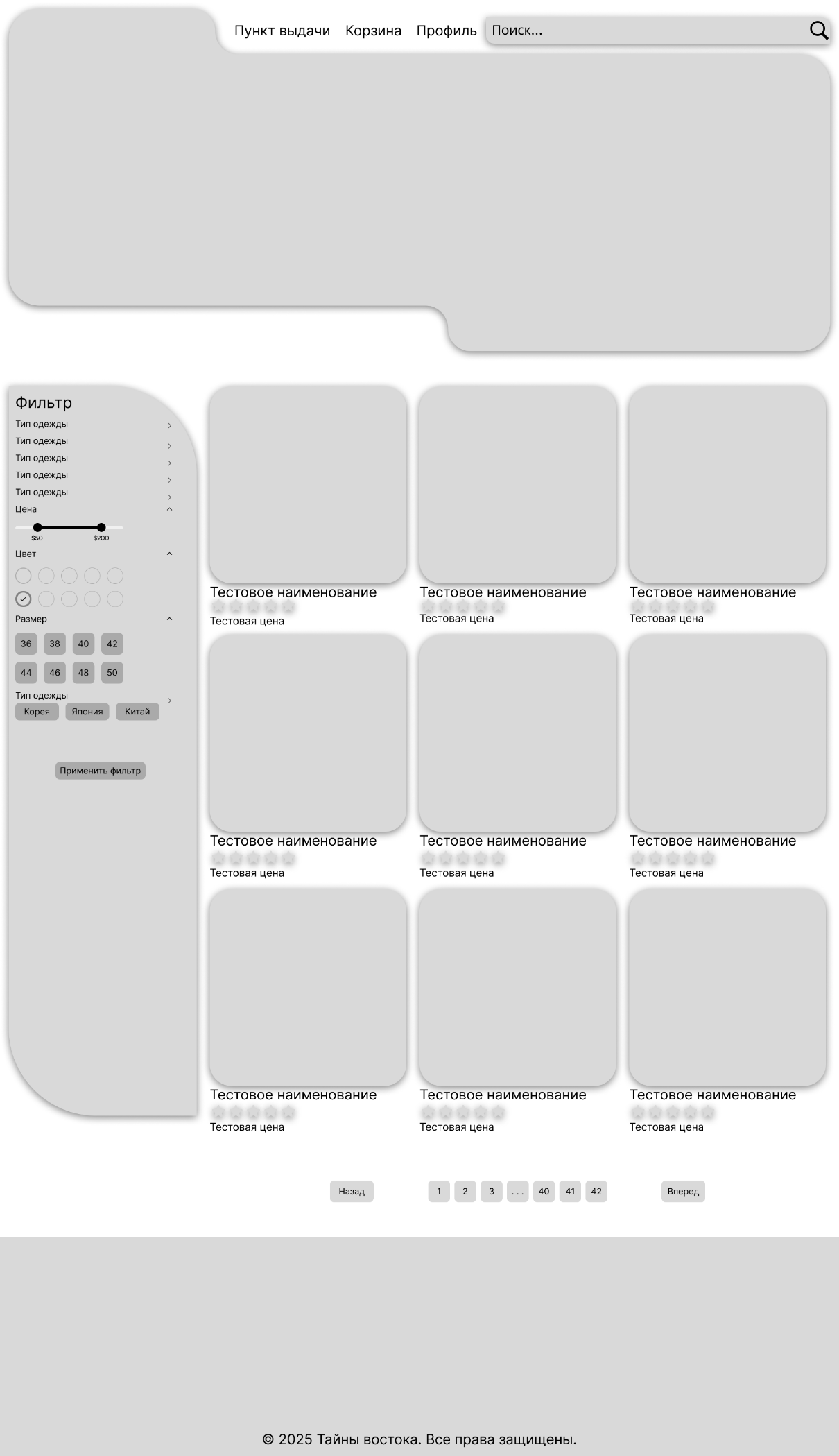


Рисунок 1 - Главная страница

**Страница товара**

При нажатии на любой товар пользователь попадает на **страницу с подробной информацией о нем**. Данный раздел включает все ключевые данные, необходимые для принятия решения о покупке.

**Основная информация на странице товара**

**Фотография товара** – основное изображение, возможно с возможностью просмотра дополнительных фото.

* **Доступные размеры** – отображаются все имеющиеся размеры товара.
* **Пол** – указано, для кого предназначен товар (мужской, женский, унисекс).
* **Доступные цвета** – выбор цветовых вариаций товара.
* **Стоимость** – актуальная цена товара.
* **Оценка товара** – средний рейтинг на основе отзывов пользователей.
* **Комментарии от покупателей** – отзывы пользователей, которые уже приобрели данный товар.

**Функциональные элементы**

* **Кнопка «Добавить в корзину»** – позволяет покупателю быстро добавить товар в корзину для дальнейшего оформления заказа.
* **Функция удаления комментариев (для менеджеров)** – у каждого отзыва есть кнопка удаления, которая доступна только менеджерам. Если в комментарии содержится **ненормативная лексика**, менеджер может удалить его.

Таким образом, **страница товара** предоставляет пользователям полную информацию о продукте, а менеджерам – инструменты для модерирования отзывов. Данные элементы интерфейса **представлены на рисунке 2.**

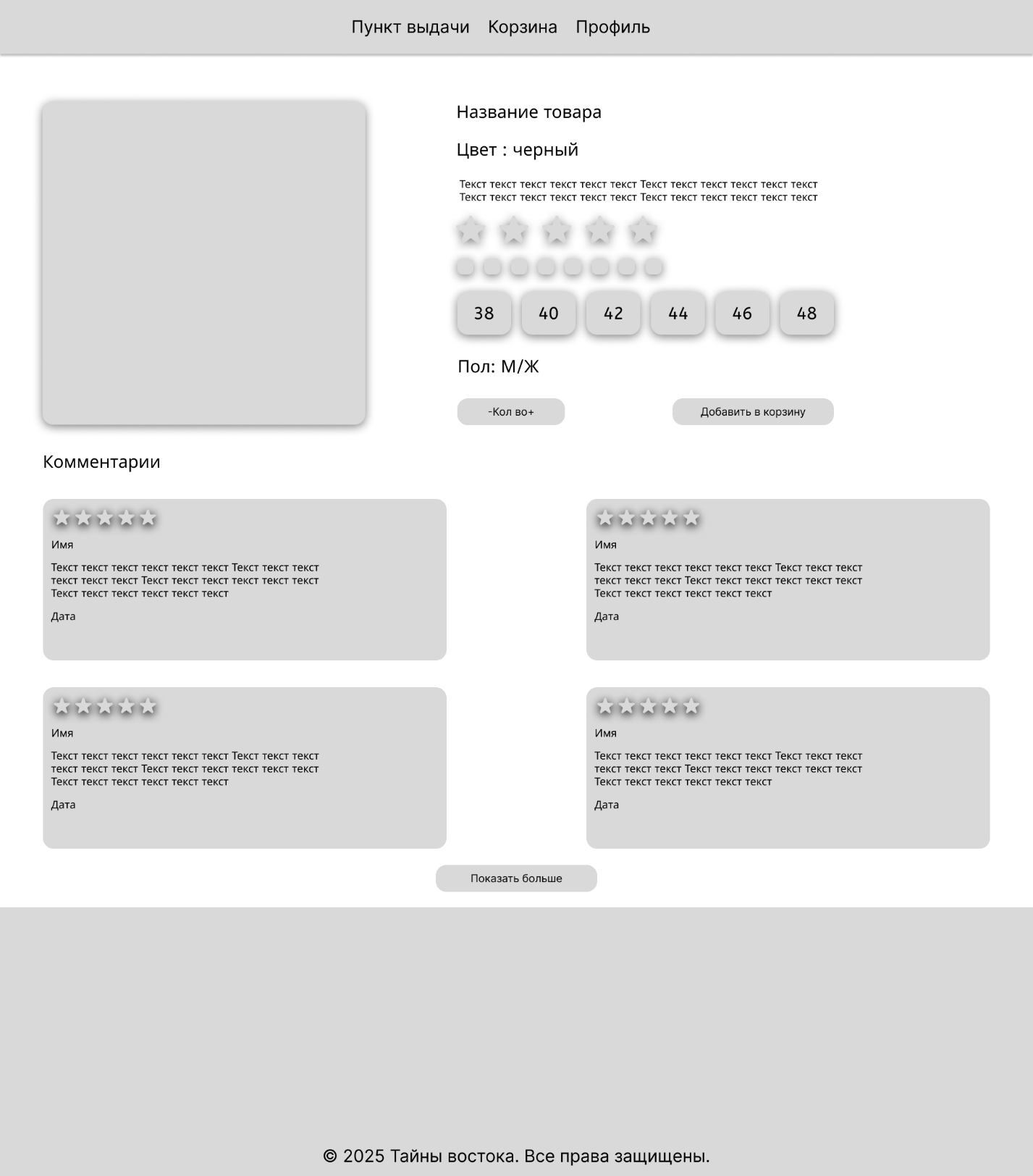


Рисунок 2 - Подробнее о товаре

**Страница корзины**

После добавления товара в корзину пользователь может перейти в **корзину**, нажав на кнопку **«Корзина»** в верхней панели навигации.

**Функционал страницы корзины**

На странице корзины отображаются:

* **Список добавленных товаров** – каждый товар с фото, названием, ценой и выбранными параметрами (размер, цвет).
* **Общее количество товаров** – отображается количество всех товаров в корзине.
* **Изменение количества товара** – возможность **увеличивать или уменьшать** количество товара.
* **Удаление товаров** – кнопка **«Удалить»** для удаления товара из корзины.
* **Выбор пункта выдачи** – позволяет выбрать место, куда будет доставлен заказ.

**Оформление заказа**

После проверки содержимого корзины пользователь нажимает **«Перейти к оформлению»**, после чего открывается **окно оплаты** (с выбором оплаты**)**, где нужно ввести данные банковской карты для завершения покупки. После успешной оплаты заказ подтверждается и обрабатывается, корзина представлена на рисунке 3.

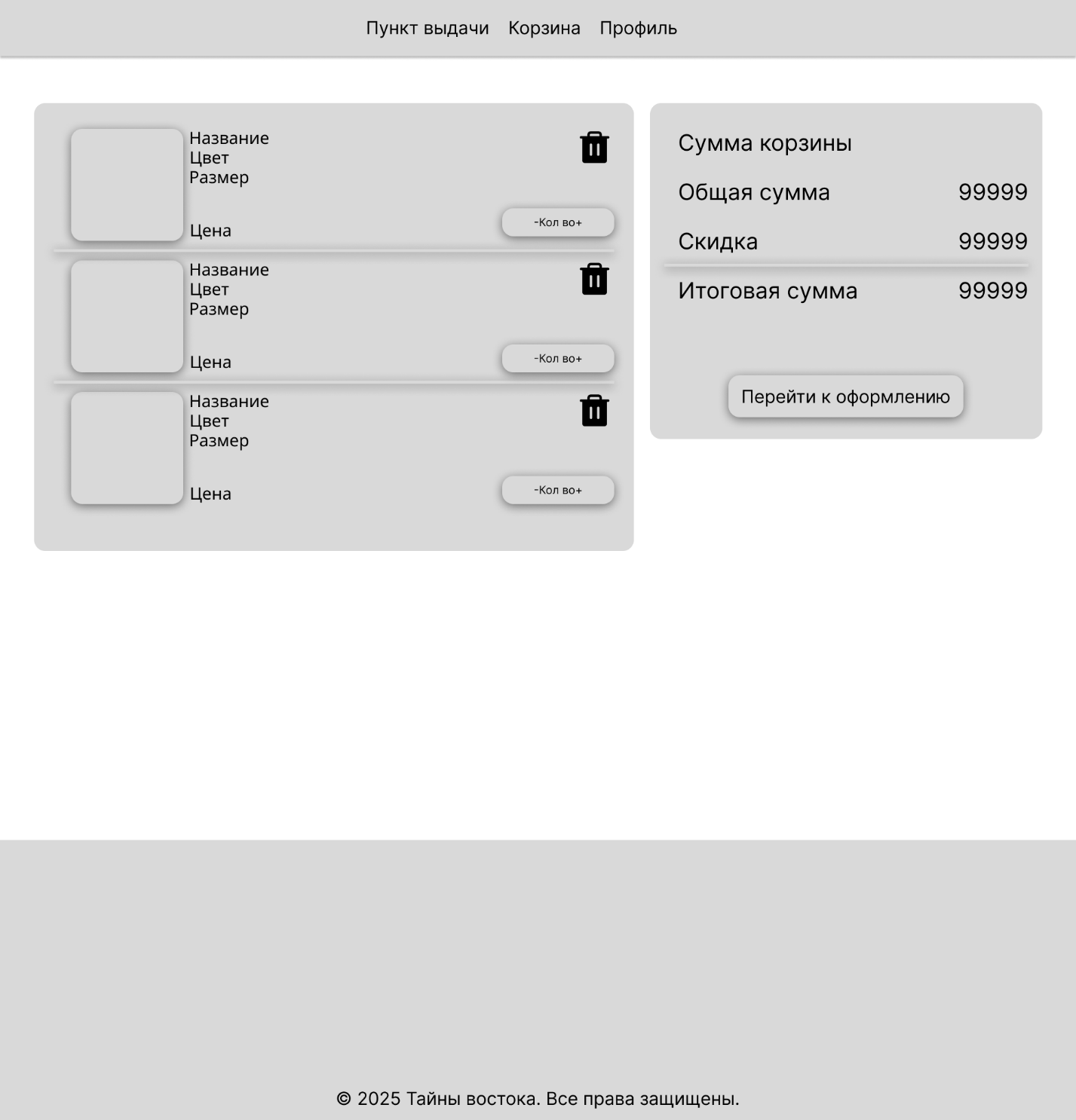


Рисунок 3 - Страница заказа

Модальное окно "Создание/Редактирование менеджера" (см. рисунок 4) содержит идентичные формы с полями для ФИО, email, пароля и телефона, а также кнопка создания, обеспечивая единый интерфейс для регистрации и изменения данных менеджеров в системе.

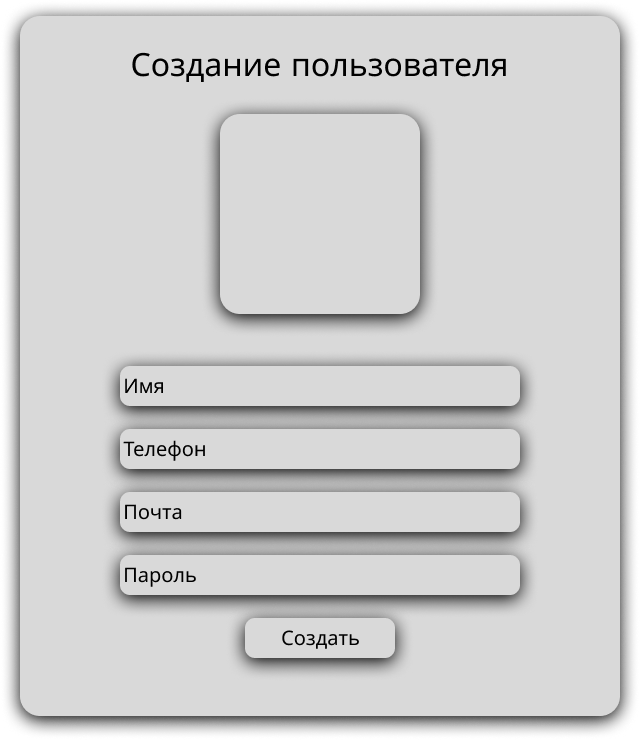


Рисунок 4- Модальное окно создания и редактирование менеджера

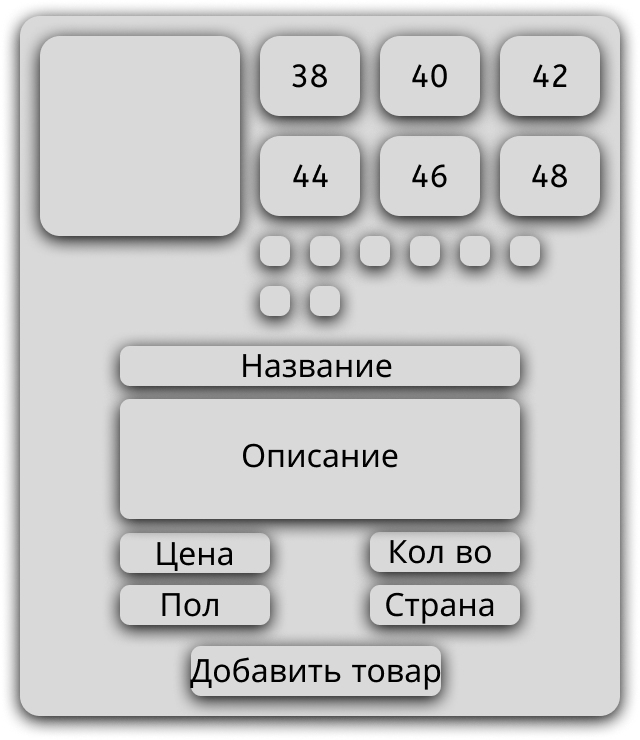
Модальное окно "Создание/редактирование товара" (см. рисунок 5) содержит форму с полями для ввода названия, описания, цены и количества товара, выбором параметров (пол, категория, размеры, цвета), загрузкой фотографии и дополнительными полями (страна производства, артикул), сохраняя единый интерфейс для добавления и изменения позиций.

Рисунок 5 – Модальное окно создания и редактирование одежды

Модальное окно "Управление заказами" (см. рисунок 6) отображает список заказов с возможностью фильтрации и изменения их статусов (в обработке, отправлен, отменен) через выпадающее меню для каждого заказа, предоставляя менеджеру инструмент для оперативного контроля выполнения заказов.

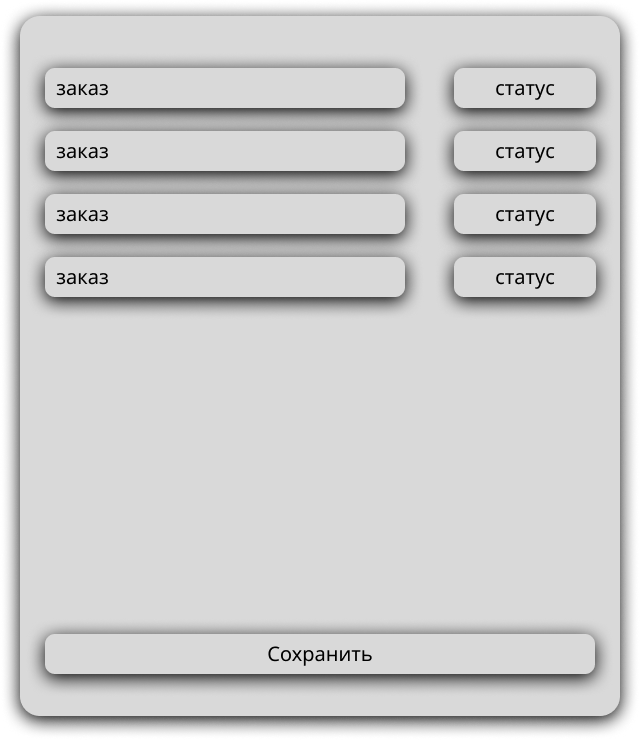


Рисунок 6 – Модальное окно для управления заказами

# ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ

## Нормализация данных

Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение.

Процесс преобразования отношений базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры БД к виду, обеспечивающему минимальную логическую избыточность, и не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение физического объёма базы данных. Конечной целью нормализации является уменьшение потенциальной противоречивости, хранимой в базе данных информации.

Устранение избыточности производится, как правило, за счёт декомпозиции отношений таким образом, чтобы в каждом отношении хранились только первичные факты.

Первая нормальная форма (1NF)

Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

Вторая нормальная форма (2NF)

Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый не ключевой атрибут неприводимо (функционально полно) зависит от её потенциального ключа. Функционально полная зависимость означает, что если потенциальный ключ является составным, то атрибут зависит от всего ключа и не зависит от его частей.

Третья нормальная форма (3NF)

Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости не ключевых атрибутов от ключевых. При проектировании БД третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

## Концептуальная модель данных

Концептуальная модель – это отражение предметной области, для которой разрабатывается база данных. Не вдаваясь в теорию, отметим, что это некая диаграмма с принятыми обозначениями элементов. Все объекты, обозначающие вещи, обозначаются в виде прямоугольника. Атрибуты, характеризующие объект - в виде овала, а связи между объектами - ромбами. Мощность связи обозначаются стрелками (в направлении, где мощность равна многим - двойная стрелка, а со стороны, где она равна единице - одинарная).

Анализируя предметную область, можно выделить следующие сущности с их атрибутами (см. рисунок 7):

1. Сущность «Пользователи» с атрибутами «Имя», «Email», «Телефон», «Пол», «Токен».
2. Сущность «Роли» с атрибутами «Название», «Код».
3. Сущность «Заказы» с атрибутами «Дата и время заказа», «Общая стоимость».
4. Сущность «Статусы» с атрибутом «Название».
5. Сущность «Пункты выдачи» с атрибутами «Город», «Улица», «Дом».
6. Сущность «Составы заказов» с атрибутами «Количество», «Общая стоимость».
7. Сущность «Товары со цветом и размером» с атрибутом «Количество».
8. Сущность «Цвета» с атрибутами «Название», «HEX».
9. Сущность «Размеры» с атрибутом «Название».
10. Сущность «Товары» с атрибутами «Фото», «Название», «Описание», «Пол», «Цена».
11. Сущность «Категории» с атрибутом «Название».
12. Сущность «Страны» с атрибутом «Название».
13. Сущность «Отзывы» с атрибутами «Оценка», «Комментарий».
14. Сущность «Корзины» с атрибутами «Количество», «Общая стоимость».

## Логическая модель данных

Рисунок 7 - Концептуальная модель данных

Логическая модель базы данных — схема базы данных, выраженная в понятиях модели данных. Этим отличается от концептуальной модели, описывающей семантику предметной области без указания технологии (конкретных методов реализации), и от физической модели, которая описывает конкретные физические механизмы, применяемые для хранения данных в накопителях.

Получившиеся логическая модель данных представлена на рисунке 8.

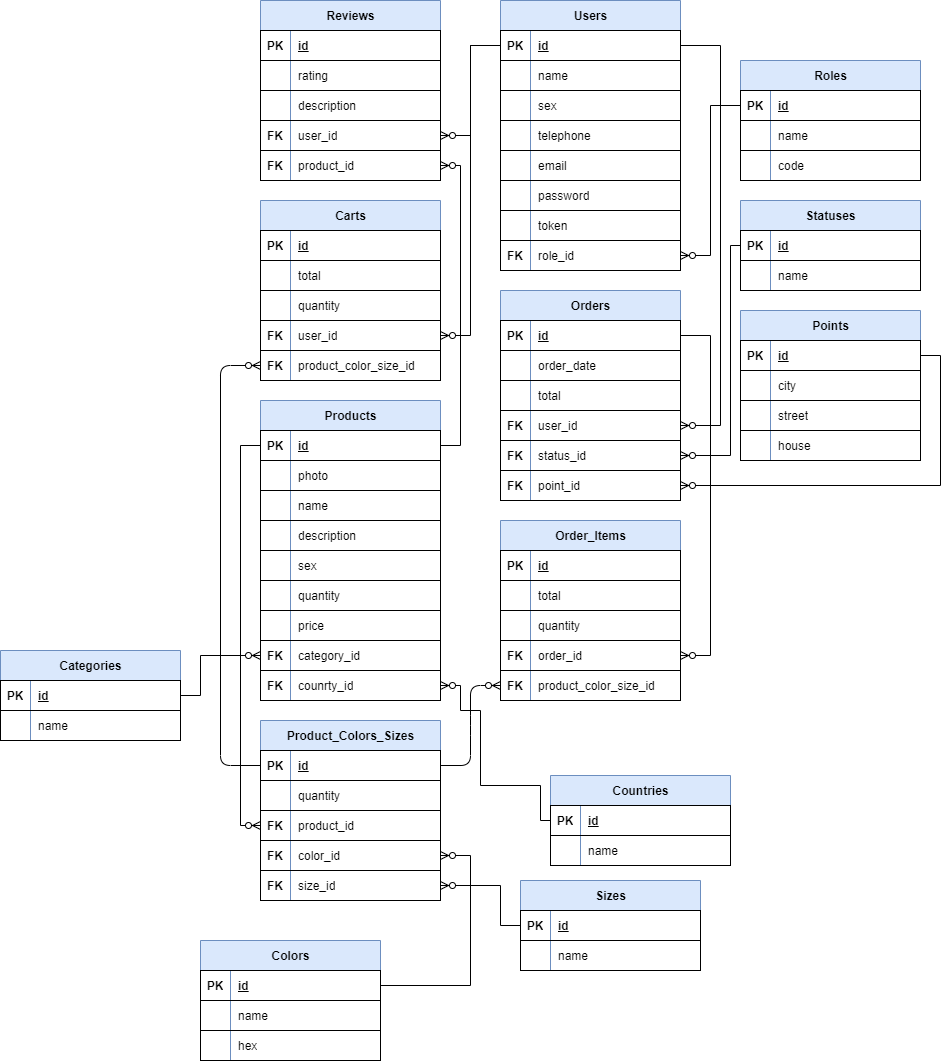


Рисунок 8 - Логическая модель данных

Физическая модель данных – это модель данных, описанная с помощью средств конкретной системы управления базами данных (СУБД). Физическая модель данных строится на базе даталогической путем добавления особенностей конкретной СУБД. К таким особенностям могут относиться поддерживаемые СУБД типы данных, соглашения о присвоении имен таблицам, атрибутам и т.д. Физическая модель данных фактически является готовым заданием на создание БД, имея которое можно реализовать БД в выбранной СУБД.

На рисунке 9 представленная физическая модель данных.

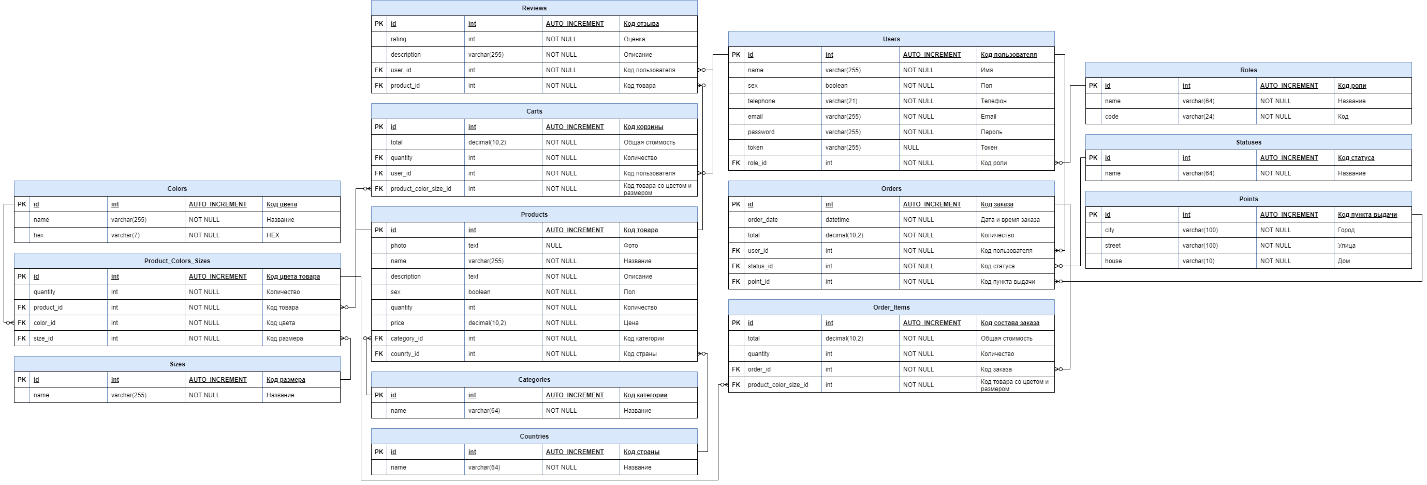


Рисунок 9 - Физическая модель данных

## Диаграмма базы данных

На рисунке 10 представлена итоговая диаграмма базы данных.

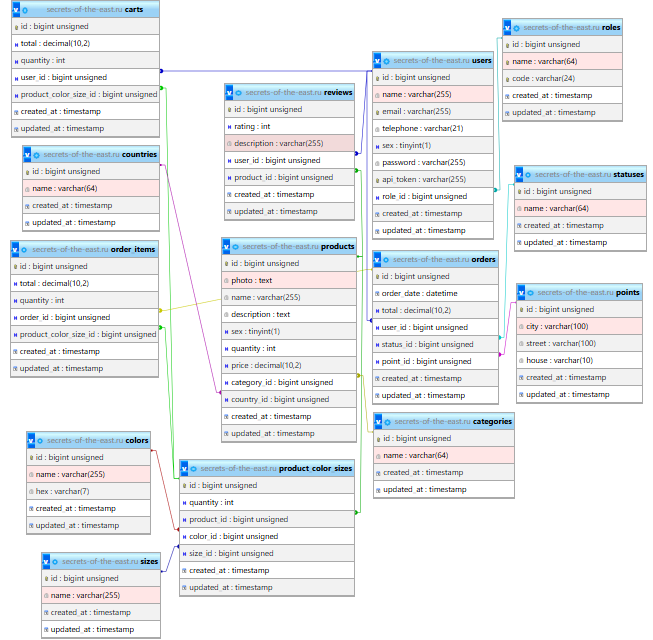


Рисунок 10 - Диаграмма базы данных

# ГЛАВА 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ФУНКЦИОНАЛА ИНФОРМАЦИОННОГО ПРОДУКТА

## Установка и настройка фреймворка Laravel12

Для установки фреймворка требуется установка Composer: Laravel использует Composer для управления зависимостями. Можно скачать и установить Composer с официального сайта.

Установка Laravel с помощью Composer: после установки Composer перейдите в терминале в каталог, в который вы хотите установить Laravel, и выполните следующую команду: composer create-project laravel/laravel .

Где (.) – директория проекта. Composer создаст новый проект Laravel в указанном каталоге.

В корневой директории сайта создаем файл .htaccess и прописываем правила перенаправления запросов в точку входа в приложение (в папку public).

RewriteEngine On  
RewriteRule ^(.\*)$ public/$1 [L]

Этот фрагмент кода предназначен для настройки правил перезаписи URL веб-сервера Apache.

RewriteEngine On: - Эта строка включает модуль перезаписи модуля mod\_rewrite в Apache.

RewriteRule ^(.\*)$ public/$1 [L] - Это правило перезаписи, которое перенаправляет все запросы на веб-сервере к корневому каталогу /public. Знак (.\*)? соответствует любой строке в URL-адресе, а $1 обозначает часть URL, совпавшую с выражением в скобках. [L] - Этот флаг означает “если это правило совпало, остановите обработку дальнейших правил перезаписи”. Это важно, чтобы избежать бесконечных циклов перезаписи. Таким образом, эта строка перенаправляет все запросы на сервере к файлам в каталоге /public.

Параметры в файле .env, можно увидеть на листинге 1, являются настройками подключения к базе данных MySQL для проекта Laravel:

***Листинг 1. Настройки подключения к базе данных***

DB\_CONNECTION=mysql - Этот параметр указывает, какой вы используете драйвер базы данных для соединения с базой данных.  
DB\_HOST=127.0.0.1 - Здесь указывается хост базы данных.  
DB\_PORT=3306 - Этот параметр указывает порт.  
DB\_DATABASE= secrets-of-the-east.ru - Это имя базы данных.  
DB\_USERNAME=root - Это имя пользователя для доступа к базе данных MySQL.  
DB\_PASSWORD= - Это пароль для пользователя базы данных MySQL.

SESSION\_DRIVER=file - Это строка конфигурации, которая определяет, какой драйвер используется для хранения данных сессий в Laravel.

## Создание миграций

Миграции в Laravel - это способ создания и изменения структуры базы данных с использованием кода PHP вместо прямых SQL-запросов. Это помогает сохранить структуру базы данных в репозитории проекта и упростить совместную работу над базой данных.

Создаем миграции для таблиц fine, statuses, roles, bonuses, users, accesses для этого в корневой директории проекта выполняем следующие команды:

php artisan make: migration create\_roles\_table

php artisan make: migration create\_users\_table

php artisan make: migration create\_statuses\_table

php artisan make: migration create\_points\_table

php artisan make: migration create\_categories\_table

php artisan make: migration create\_countries\_table

php artisan make: migration create\_colors\_table

php artisan make: migration create\_sizes\_table

php artisan make: migration create\_products\_table

php artisan make: migration create\_reviews\_table

php artisan make: migration create\_orders\_table

php artisan make: migration create\_product\_color\_sizes\_table

php artisan make: migration create\_carts\_table

php artisan make: migration create\_order\_items\_table

Миграции похожи на контроль версий для вашей базы данных, позволяют вашей команде определять схемы базы данных приложения и совместно использовать их определение.

Фасад Schema обеспечивает независимую от базы данных поддержку для создания и управления таблицами во всех поддерживаемых Laravel системах баз данных. В обычной ситуации, этот фасад используется для создания и изменения таблиц / столбцов базы данных во время миграции. Итоговые миграции представлены в листинге 2.

**Листинг 2. Миграции базы данных**

Schema::create('roles', function (Blueprint $table) {  
 $table->id();  
 $table->string('name', 64)->unique();  
 $table->string('code', 24)->unique();  
 $table->timestamps();  
});

Schema::create('users', function (Blueprint $table) {  
 $table->id();  
 $table->string('name',255);  
 $table->string('email', 255)->unique();  
 $table->string('telephone', 21);  
 $table->boolean('sex');  
 $table->string('password',255);  
 $table->string('api\_token', 255)->nullable()->unique();  
 $table->foreignId('role\_id')->constrained();  
 $table->timestamps();  
});

Schema::create('statuses', function (Blueprint $table) {  
 $table->id();  
 $table->string('name', 64);  
 $table->timestamps();  
});

Schema::create('points', function (Blueprint $table) {  
 $table->id();  
 $table->string('city', 100);  
 $table->string('street', 100);  
 $table->string('house', 10);  
 $table->timestamps();  
});

Schema::create('categories', function (Blueprint $table) {  
 $table->id();  
 $table->string('name', 64);  
 $table->timestamps();  
});

Schema::create('countries', function (Blueprint $table) {  
 $table->id();  
 $table->string('name', 64);  
 $table->timestamps();  
});

Schema::create('colors', function (Blueprint $table) {  
 $table->id();  
 $table->string('name', 255);  
 $table->string('hex', 7);  
 $table->timestamps();  
});

Schema::create('sizes', function (Blueprint $table) {  
 $table->id();  
 $table->string('name', 255);  
 $table->timestamps();  
});

Schema::create('products', function (Blueprint $table) {  
 $table->id();  
 $table->text('photo')->nullable();  
 $table->string('name', 255);  
 $table->text('description');  
 $table->boolean('sex');  
 $table->integer('quantity');  
 $table->decimal('price', 10, 2);  
 $table->foreignId('category\_id')->constrained();  
 $table->foreignId('country\_id')->constrained();  
 $table->timestamps();  
});

Schema::create('reviews', function (Blueprint $table) {  
 $table->id();  
 $table->integer('rating');  
 $table->string('description', 255);  
 $table->foreignId('user\_id')->constrained();  
 $table->foreignId('product\_id')->constrained();  
 $table->timestamps();  
});

Schema::create('orders', function (Blueprint $table) {  
 $table->id();  
 $table->dateTime('order\_date');  
 $table->decimal('total', 10, 2);  
 $table->foreignId('user\_id')->constrained();  
 $table->foreignId('status\_id')->constrained();  
 $table->foreignId('point\_id')->constrained();  
 $table->timestamps();  
});

Schema::create('product\_color\_sizes', function (Blueprint $table) {  
 $table->id();  
 $table->integer('quantity');  
 $table->foreignId('product\_id')->constrained();  
 $table->foreignId('color\_id')->constrained();  
 $table->foreignId('size\_id')->constrained();  
 $table->timestamps();  
});

Schema::create('carts', function (Blueprint $table) {  
 $table->id();  
 $table->decimal('total', 10, 2);  
 $table->integer('quantity');  
 $table->foreignId('user\_id')->constrained();  
 $table->foreignId('product\_color\_size\_id')->constrained();  
 $table->timestamps();  
});

Schema::create('order\_items', function (Blueprint $table) {  
 $table->id();  
 $table->decimal('total', 10, 2);  
 $table->integer('quantity');  
 $table->foreignId('order\_id')->constrained();  
 $table->foreignId('product\_color\_size\_id')->constrained();  
 $table->timestamps();  
});

Выполняем миграцию с помощью команды: php artisan migrate.

## Создание исключений

Во время выбрасывания исключений Laravel проверяет, есть ли в классе исключения метод render(), если да, то он использует метод этого исключения для отображения результата. Если вы не хотите полагаться на глобальную систему отлова исключений Laravel, то можете вернуть ответ в JSON напрямую из контроллера.

Illuminate\Http\Exceptions\HttpResponseException - это особое исключение, встроенное в Laravel. Особенность этого исключения заключается в том, что оно уже содержит шаблон ответа для клиента, поэтому Laravel просто возвращает ответ из этого исключения.

Созданный класс исключения ApiException расширяет базовый класс Exception. В конструкторе он принимает код и сообщение исключения, а также, при необходимости, массив ошибок. При создании объекта исключения он вызывает конструктор родительского класса Exception, передавая ему сообщение и код исключения.

Создадим особое исключение app\Exceptions\Api\ApiException.php, можно увидеть на листинге 3, которое будет определять шаблон выдаваемых пользователю ошибок (будь то ошибка авторизации или ошибка валидации входных данных, либо ошибка доступа к определенному функционалу приложения) со следующим содержимым:

Конструктор принимает три параметра:

$message: Сообщение об ошибке (строка).

$code: HTTP-статус код (по умолчанию 500).

$errors: Дополнительные ошибки (например, ошибки валидации).

Создается массив $response, который содержит:

code: Статус-код.

message: Сообщение об ошибке.

errors: Ошибки, если они переданы.

Затем вызывается конструктор родительского класса (HttpResponseException) с JSON-ответом, установленным на указанный статус-код.

***Листинг 3. Класс исключения ApiException***

class ApiException extends HttpResponseException  
{  
 public function \_\_construct(string $message, int $code = 500, $errors = [])  
 {  
 $response = [  
 'code' => $code,  
 'message' => $message,  
 ];  
 if (!empty($errors)) {  
 $response['errors'] = $errors;  
 }  
 parent::*\_\_construct*( response()->json($response)->setStatusCode($code));  
 }  
}

## Создание моделей

Модели в фреймворке Laravel играют ключевую роль в работе с данными вашего приложения, например:

* Работа с данными базы данных: Модели представляют собой объектно-ориентированный способ взаимодействия с таблицами базы данных. Каждая модель обычно связана с определенной таблицей в базе данных, и она предоставляет удобные методы для выполнения запросов к этой таблице, таких как создание, чтение, обновление и удаление записей.
* Организация бизнес-логики: Модели часто содержат бизнес-логику вашего приложения, связанную с данными. Например, методы модели могут включать валидацию данных, вычисления или другие операции, связанные с обработкой данных.
* Отношения между таблицами: В Laravel отношения между таблицами базы данных могут быть легко определены в моделях. Это позволяет вам работать с данными, которые связаны между собой, например, получать связанные записи или устанавливать связи между ними.
* Использование в контроллерах и представлениях: Модели обычно используются в контроллерах для выполнения операций с данными, а также в представлениях для отображения этих данных. Они обеспечивают удобный способ доступа к данным вашего приложения из любой его части.
* Тестирование: Использование моделей позволяет упростить процесс тестирования вашего приложения. Вы можете создавать экземпляры моделей в тестах и проверять их поведение в различных сценариях.

Чтобы сгенерировать новую модель Eloquent, необходимо использовать команду:

php artisan make:model.

Создадим модели ко всем таблицам нашей базы данных, для этого поочередно выполняем следующие команды:

php artisan make:model Cart

php artisan make:model Category

php artisan make:model Color

php artisan make:model Country

php artisan make:model Order

php artisan make:model OrderItem

php artisan make:model Point

php artisan make:model Product

php artisan make:model ProductColorSize

php artisan make:model Review

php artisan make:model Role

php artisan make:model Size

php artisan make:model Status

php artisan make:model User

Далее отредактируем наши модели, можно увидеть на листинге 4, указав связи:

***Листинг 4. Модели базы данных***

class Cart extends Model  
{  
 protected $fillable = ['total', 'quantity', 'user\_id', 'product\_color\_size\_id'];  
  
 public function order()  
 {  
 return $this->belongsTo(Order::class);  
 }  
 public function productColorSize()  
 {  
 return $this->belongsTo(ProductColorSize::class);  
 }  
}

class Category extends Model  
{  
 protected $fillable = ['name'];  
  
 public function products(){  
 return $this->hasMany(Product::class);  
 }  
}

class Color extends Model  
{  
 protected $fillable = ['name', 'hex'];  
  
 public function productColorSizes(){  
 return $this->hasMany(ProductColorSize::class);  
 }  
}

class Country extends Model  
{  
 protected $fillable = ['name'];  
  
 public function products(){  
 return $this->hasMany(Product::class);  
 }  
}

class Order extends Model  
{  
 protected $fillable = ['order\_date', 'total', 'user\_id', 'status\_id', 'point\_id'];  
  
 public function user()  
 {  
 return $this->belongsTo(User::class);  
 }  
 public function status()  
 {  
 return $this->belongsTo(Status::class);  
 }  
 public function point()  
 {  
 return $this->belongsTo(Point::class);  
 }  
 public function orderItems(){  
 return $this->hasMany(OrderItem::class);  
 }  
}

class OrderItem extends Model  
{  
 protected $fillable = ['total', 'quantity', 'order\_id', 'product\_color\_size\_id'];  
  
 public function order()  
 {  
 return $this->belongsTo(Order::class);  
 }  
 public function productColorSize()  
 {  
 return $this->belongsTo(ProductColorSize::class);  
 }  
}

class Point extends Model  
{  
 protected $fillable = ['city', 'street', 'house'];  
  
 public function orders(){  
 return $this->hasMany(Order::class);  
 }  
}

class Product extends Model  
{  
 protected $fillable = ['photo', 'name', 'description', 'sex', 'quantity', 'price', 'category\_id', 'country\_id'];  
  
 public function category()  
 {  
 return $this->belongsTo(Category::class);  
 }  
 public function country()  
 {  
 return $this->belongsTo(Country::class);  
 }  
  
 public function productColorSizes()  
 {  
 return $this->hasMany(ProductColorSize::class);  
 }  
 public function reviews()  
 {  
 return $this->hasMany(Review::class);  
 }  
}

class ProductColorSize extends Model  
{  
 protected $fillable = ['quantity', 'product\_id', 'color\_id', 'size\_id'];  
  
 public function product()  
 {  
 return $this->belongsTo(Product::class);  
 }  
 public function color()  
 {  
 return $this->belongsTo(Color::class);  
 }  
 public function size()  
 {  
 return $this->belongsTo(Size::class);  
 }  
 public function orderItems()  
 {  
 return $this->hasMany(OrderItem::class);  
 }  
 public function carts()  
 {  
 return $this->hasMany(Cart::class);  
 }  
}

class Review extends Model  
{  
 protected $fillable = ['rating', 'description', 'user\_id', 'product\_id'];  
  
 public function user()  
 {  
 return $this->belongsTo(User::class);  
 }  
 public function product()  
 {  
 return $this->belongsTo(Product::class);  
 }  
}

class Role extends Model  
{  
 protected $fillable = ['name', 'code'];  
  
 public function users(){  
 return $this->hasMany(User::class);  
 }  
}

class Size extends Model  
{  
 protected $fillable = ['name'];  
  
 public function productColorSizes(){  
 return $this->hasMany(ProductColorSize::class);  
 }  
}

class Status extends Model  
{  
 protected $fillable = ['name'];  
  
 public function orders(){  
 return $this->hasMany(Order::class);  
 }  
}

class User extends *Authenticatable*{  
 use HasApiTokens, HasFactory, Notifiable;  
  
 protected $fillable = [  
 'name', 'email', 'telephone', 'sex', 'password', 'api\_token', 'role\_id',  
 ];  
 protected $hidden = [  
 'password',  
 'api\_token',  
 ];  
 protected function casts(): array  
 {  
 return [  
 'password' => 'hashed',  
 ];  
 }  
 public function role()  
 {  
 return $this->belongsTo(Role::class);  
 }  
 public function orders()  
 {  
 return $this->hasMany(Order::class);  
 }  
 public function reviews(){  
 return $this->hasMany(Review::class);  
 }  
}

Класс User содержит поля $fillable и представляет собой модель пользователя в Laravel, которая использует трейты HasApiTokens, HasFactory и Notifiable и связана с моделями Role, Fine, Access, Status.

С помощью $hidden определяются, атрибуты, которые должны быть скрыты при сериализации модели в JSON.

С помощью метода casts определяются, атрибуты модели, которые должны быть приведены к определенным типам. В нашем случае поле password автоматически кэшируется при сохранении в базу данных.

## Создание контроллеров

В фреймворке Laravel контроллеры играют важную роль в организации логики приложения. Они обрабатывают HTTP-запросы и возвращают соответствующие HTTP-ответы.

Выполним генерация контроллеров, необходимых для решения технического задания, для этого выполним следующие команды:

php artisan make:controller AuthController

php artisan make:controller CategoryController

php artisan make:controller ColorController

php artisan make:controller FilterController

php artisan make:controller OrderController

php artisan make:controller PointController

php artisan make:controller ProductController

php artisan make:controller ProfileController

php artisan make:controller ReviewController

php artisan make:controller SizeController

php artisan make:controller StatusController

php artisan make:controller UserController

php artisan make:controller ViewFilterController

php artisan make:controller CartController

Далее отредактируем наши котроллеры, можно увидеть на листингах 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, написав необходимые методы:

***Листинг 5. Контроллер AuthController для авторизации***

class AuthController  
{  
 // Регистрация пользователя  
 public function register(RegisterRequest $request)  
 {  
 // Извлекаем role\_id для роли 'Пользователь'  
 $role\_id = Role::*where*('code', 'user')->first()->id;  
  
 // Извлекаем валидированные данные  
 $validated = $request->validated();  
  
 // Создаем нового пользователя  
 $user = User::*create*([  
 ...$validated,  
 'role\_id' => $role\_id]);  
  
 // Создание токена для пользователя  
 $user->api\_token = Hash::make(Str::*random*(60));  
 $user->save();  
  
 // Возвращаем ответ с токеном  
 return response()->json([  
 'user' => $user,  
 'token' => $user->api\_token  
 ])->setStatusCode(201);  
 }  
 // Авторизация  
 public function login(Request $request) {  
 if (!Auth::attempt($request->only('email', 'password'))) {  
 throw new ApiException("Неверная почта или пароль", 401);  
 }  
 // Получение текущего пользователя  
 $user = Auth::user();  
 // Создание нового токена для пользователя  
 $user->api\_token = Hash::make(Str::*random*(60));  
 $user->save();  
 return response()->json(['token' => $user->api\_token, 'user' => $user])->setStatusCode(200);  
 }  
 // Выход  
 public function logout(Request $request) {  
 // Получение текущего пользователя  
 $user = Auth::user();  
 // Создание нового токена для пользователя  
 $user->api\_token = null;  
 $user->save();  
 return response()->json(['message' => 'Вы вышли из системы'])->setStatusCode(200);  
 }  
}

***Листинг 6. Контроллер CategoryController для категорий***

class CategoryController  
{  
 public function store(CreateCategoryRequest $request) {  
  
 if (Auth::user()->role->code != 'admin') {  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 $validated = $request->validated();  
 // Создание нового адреса  
 $category = Category::*create*([  
 ...$validated,  
 'name' => $request->name,  
 ]);  
  
 // Возвращаем ответ с созданным объектом и статусом 201 (создано)  
 return response()->json($category, 201);  
 }  
}

***Листинг 7. Контроллер ColorController для цветов***

class ColorController  
{  
 public function index()  
 {  
 $colors = Color::*all*();  
 if ($colors->isEmpty()) {  
 throw new ApiException( 'Не найдено', 404);  
 }  
 return response()->json($colors)->setStatusCode(200);  
 }  
}

***Листинг 8. Контроллер FilterController для фильтров по параметрам или по названию, описанию***

class FilterController  
{  
 public function filterProducts(FilterRequest $request)  
 {  
 // Валидация входных параметров  
 $validated = $request->validated();  
  
 // Начинаем построение запроса  
 $query = Product::*query*()->with(['category', 'country']);  
  
 // Применяем фильтры  
 if ($request->has('category\_id')) {  
 $category = Category::*find*($request->category\_id);  
 if (!$category) {  
 throw new ApiException('Категория не найдена', 404);  
 }  
 $query->where('category\_id', $request->category\_id);  
 }  
  
 if ($request->has('country\_id')) {  
 $country = Country::*find*($request->country\_id);  
 if (!$country) {  
 throw new ApiException('Страна не найдена', 404);  
 }  
 $query->where('country\_id', $request->country\_id);  
 }  
  
 if ($request->has('min\_price')) {  
 $query->where('price', '>=', $request->min\_price);  
 }  
  
 if ($request->has('max\_price')) {  
 $query->where('price', '<=', $request->max\_price);  
 }  
  
 if ($request->has('sex')) {  
 $query->where('sex', $request->sex);  
 }  
  
 // Получаем и возвращаем результаты  
 $products = $query->get();  
  
 if ($products->isEmpty()) {  
 throw new ApiException('Товары по заданным фильтрам не найдены', 404);  
 }  
  
 return response()->json([  
 'success' => true,  
 'filters' => $validated,  
 'count' => $products->count(),  
 'data' => $products  
 ]);  
 }  
 // Поиск товаров по названию или описанию  
 public function search(Request $request)  
 {  
 $request->validate([  
 'q' => 'nullable|string|max:255',  
 ]);  
  
 $search = $request->input('q');  
  
 $query = Product::*with*(['category', 'country']);  
  
 if ($search) {  
 $query->where(function ($q) use ($search) {  
 $q->where('name', 'like', "%{$search}%")  
 ->orWhere('description', 'like', "%{$search}%");  
 });  
 }  
  
 $products = $query->get();  
  
 return response()->json([  
 'success' => true,  
 'query' => $search,  
 'count' => $products->count(),  
 'data' => $products,  
 ], 200);  
 }  
}

***Листинг 9. Контроллер OrderController для заказов***

class OrderController  
{  
 // Получение списка всех заказов пользователя  
 public function index()  
 {  
 try {  
 $user = auth()->user();  
 $isAdmin = $user->role->code === 'admin' || $user->role->code === 'manager';  
  
 // Загружаем заказы с нужными связями  
 $orders = $isAdmin  
 ? Order::*with*([  
 'user', 'point', 'status',  
 'orderItems.productColorSize.product',  
 'orderItems.productColorSize.color',  
 'orderItems.productColorSize.size'  
 ])->get()  
 : $user->orders()  
 ->with([  
 'point', 'status',  
 'orderItems.productColorSize.product',  
 'orderItems.productColorSize.color',  
 'orderItems.productColorSize.size'  
 ])->get();  
  
 // Если заказов нет - возвращаем 404  
 if ($orders->isEmpty()) {  
 return response()->json([  
 'status' => 'error',  
 'message' => $isAdmin  
 ? 'В системе пока нет заказов'  
 : 'У вас пока нет заказов'  
 ], 404);  
 }  
  
 // Трансформируем заказы  
 $ordersTransformed = $orders->map(function ($order) use ($isAdmin) {  
 $totalCost = $order->orderItems->sum(function ($item) {  
 return $item->productColorSize->product->price \* $item->quantity;  
 });  
  
 return [  
 'id' => $order->id,  
 'order\_date' => $order->order\_date,  
 'status' => $order->status->name ?? null,  
 'total\_cost' => $totalCost,  
 'user' => $isAdmin ? $order->user->name : null,  
 'address' => [  
 'city' => $order->point->city ?? null,  
 'street' => $order->point->street ?? null,  
 'house' => $order->point->house ?? null,  
 ],  
 'items\_count' => $order->orderItems->count()  
 ];  
 });  
  
 return response()->json([  
 'status' => 'success',  
 'data' => $ordersTransformed  
 ]);  
  
 } catch (\Exception $e) {  
 return response()->json([  
 'status' => 'error',  
 'message' => 'Произошла ошибка при получении списка заказов'  
 ], 500);  
 }  
 }  
  
 // Создание нового заказа с тестовой оплатой  
 public function store(Request $request)  
 {  
 $clientId = $request->user()->id;  
 $addressId = $request->input('point\_id');  
  
 // Проверка наличия адреса  
 if (!$addressId) {  
 return response()->json([  
 'status' => 'error',  
 'message' => 'Не указан адрес доставки.',  
 ], 400);  
 }  
  
 // Загружаем корзину с нужными связями  
 $cartProducts = Cart::*where*('user\_id', $clientId)  
 ->with('productColorSize.product')  
 ->get();  
  
 if ($cartProducts->isEmpty()) {  
 return response()->json([  
 'status' => 'error',  
 'message' => 'Корзина пуста. Добавьте товары для оформления заказа.',  
 ], 400);  
 }  
  
 // Расчёт общей стоимости  
 $totalCost = $cartProducts->sum(function ($item) {  
 return $item->productColorSize->product->price \* $item->quantity;  
 });  
  
 try {  
 Stripe::*setApiKey*(env('STRIPE\_SECRET'));  
  
 // 1. Создаем сессию оплаты Stripe Checkout  
 $session = Session::*create*([  
 'payment\_method\_types' => ['card'],  
 'line\_items' => [[  
 'price\_data' => [  
 'currency' => 'rub',  
 'product\_data' => [  
 'name' => 'Заказ от ' . $request->user()->name,  
 ],  
 'unit\_amount' => $totalCost \* 100,  
 ],  
 'quantity' => 1,  
 ]],  
 'mode' => 'payment',  
 'success\_url' => route('checkout.success') . '?session\_id={CHECKOUT\_SESSION\_ID}',  
 'cancel\_url' => route('checkout.cancel'),  
 'metadata' => [  
 'user\_id' => $clientId,  
 'address\_id' => $addressId,  
 // Передаем только необходимые минимальные данные  
 'cart\_summary' => json\_encode([  
 'items\_count' => $cartProducts->count(),  
 'total' => $totalCost  
 ])  
 ],  
 ]);  
  
 // 2. Возвращаем URL для перенаправления на страницу оплаты Stripe  
 return response()->json([  
 'status' => 'payment\_required',  
 'message' => 'Требуется подтверждение оплаты',  
 'payment\_url' => $session->url,  
 'session\_id' => $session->id  
 ], 200);  
  
 } catch (\Exception $e) {  
 return response()->json([  
 'status' => 'error',  
 'message' => 'Ошибка при создании сессии оплаты: ' . $e->getMessage(),  
 ], 500);  
 }  
 }  
  
 public function handleSuccess(Request $request)  
 {  
 try {  
 Stripe::*setApiKey*(env('STRIPE\_SECRET'));  
 $sessionId = $request->get('session\_id');  
 $session = Session::*retrieve*($sessionId);  
  
 // Получаем данные из metadata  
 $clientId = $session->metadata->user\_id;  
 $addressId = $session->metadata->address\_id;  
  
 // 1. Получаем корзину пользователя из базы данных  
 $cartProducts = Cart::*where*('user\_id', $clientId)  
 ->with('productColorSize.product')  
 ->get();  
  
 if ($cartProducts->isEmpty()) {  
 throw new \Exception('Корзина пуста или уже обработана');  
 }  
  
 // 2. Создаем заказ  
 $order = Order::*create*([  
 'user\_id' => $clientId,  
 'point\_id' => $addressId,  
 'order\_date' => now(),  
 'total' => $session->amount\_total / 100,  
 'status\_id' => 2,  
 'payment\_id' => $session->payment\_intent,  
 'payment\_status' => 'succeeded',  
 ]);  
  
 // 3. Добавляем товары в заказ  
 foreach ($cartProducts as $item) {  
 OrderItem::*create*([  
 'order\_id' => $order->id,  
 'product\_color\_size\_id' => $item->product\_color\_size\_id,  
 'quantity' => $item->quantity,  
 'total' => $item->productColorSize->product->price \* $item->quantity,  
 ]);  
 }  
  
 // 4. Очищаем корзину  
 Cart::*where*('user\_id', $clientId)->delete();  
  
 return response()->json([  
 'status' => 'success',  
 'message' => 'Заказ успешно оформлен и оплачен',  
 'order\_id' => $order->id,  
 ], 201);  
  
 } catch (\Exception $e) {  
 return response()->json([  
 'status' => 'error',  
 'message' => 'Ошибка при обработке платежа: ' . $e->getMessage(),  
 ], 500);  
 }  
 }  
  
 public function handleCancel(Request $request)  
 {  
 return response()->json([  
 'status' => 'canceled',  
 'message' => 'Оплата отменена',  
 ], 200);  
 }  
  
 public function show($id)  
 {  
 $user = auth()->user();  
 $isAdmin = $user->role->code === 'admin' || $user->role->code === 'manager';  
  
 // Загружаем заказ с вложенными связями  
 $order = Order::*with*([  
 'user',  
 'point',  
 'status',  
 'orderItems.productColorSize.product',  
 'orderItems.productColorSize.color',  
 'orderItems.productColorSize.size'  
 ])  
 ->when(!$isAdmin, function ($query) use ($user) {  
 // Ограничиваем по пользователю, если не админ  
 $query->where('user\_id', $user->id);  
 })  
 ->find($id);  
  
 if (!$order) {  
 return response()->json(['error' => 'Заказ не найден'], 404);  
 }  
  
 // Список товаров  
 $items = $order->orderItems->map(function ($item) {  
 $pcs = $item->productColorSize;  
 $product = $pcs->product;  
  
 return [  
 'product\_id' => $product->id,  
 'photo' => $product->photo,  
 'name' => $product->name,  
 'price' => $product->price,  
 'quantity' => $item->quantity,  
 'color' => $pcs->color->name ?? null,  
 'size' => $pcs->size->name ?? null,  
 'total' => round($product->price \* $item->quantity, 2),  
 ];  
 });  
  
 $totalCost = $items->sum('total');  
  
 return response()->json([  
 'id' => $order->id,  
 'order\_date' => $order->order\_date,  
 'status' => $order->status->name ?? null,  
 'total\_cost' => $totalCost,  
 'user' => $isAdmin ? $order->user->name : null,  
 'address' => [  
 'city' => $order->point->city ?? null,  
 'street' => $order->point->street ?? null,  
 'house' => $order->point->house ?? null,  
 ],  
 'items' => $items,  
 ]);  
 }  
  
 // Обновление заказа (статус)  
 public function update(Request $request, $id)  
 {  
 if (Auth::user()->role->code != 'admin' && 'manager') {  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 // Загружаем заказ  
 $order = Order::*find*($id);  
  
 if (!$order) {  
 return response()->json(['error' => 'Заказ не найден'], 404);  
 }  
  
 // Валидируем данные  
 $validated = $request->validate([  
 'status\_id' => 'nullable|integer',  
 ]);  
  
 $order->update($validated);  
  
 return response()->json([  
 'message' => 'Заказ обновлён',  
 'order' => $order,  
 ]);  
 }  
  
 // Отмена заказа  
 public function destroy($id)  
 {  
 $user = auth()->user();  
  
 // Находим заказ текущего пользователя  
 $order = $user->orders()->find($id);  
  
 if (!$order) {  
 return response()->json(['error' => 'Заказ не найден'], 404);  
 }  
  
 $order->status\_id = 4;  
 $order->save();  
  
 return response()->json(['message' => 'Заказ отменён']);  
 }  
}

***Листинг 10. Контроллер PointController для пунктов выдачи***

class PointController  
{  
 // Получение списка всех адресов  
 public function index()  
 {  
 $points = Point::*all*();  
 if ($points->isEmpty()) {  
 throw new ApiException( 'Не найдено', 404);  
 }  
 return response()->json($points)->setStatusCode(200);  
 }  
  
 // Создание нового адреса  
 public function store(CreatePointRequest $request)  
 {  
 if(Auth::user()->role->code != 'admin'){  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
 $validated = $request->validated();  
 // Создание нового адреса  
 $point = Point::*create*([  
 ...$validated,  
 'city' => $request->city,  
 'street' => $request->street,  
 'house' => $request->house,  
 ]);  
  
 // Возвращаем ответ с созданным объектом и статусом 201 (создано)  
 return response()->json($point, 201);  
 }  
  
 // Обновление данных адреса  
 public function update(UpdatePointRequest $request, $id)  
 {  
 if (Auth::user()->role->code != 'admin') {  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 $point = Point::*find*($id);  
 if (!$point) {  
 return response()->json(['error' => 'Пункт выдачи не найден'], 404);  
 }  
 $validated = $request->validated();  
 // Обновляем данные адреса  
 $point->update($validated);  
  
 // Возвращаем обновленный адрес  
 return response()->json($point);  
 }  
   
 // Удаление адреса по id  
 public function destroy($id)  
 {  
 if (Auth::user()->role->code != 'admin') {  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 $point = Point::*find*($id);  
  
 if (!$point) {  
 return response()->json(['message' => 'Пункт выдачи не найден'], 404);  
 }  
  
 // Проверка на наличие связанных записей  
 if (  
 $point->orders()->exists()  
 // добавь сюда другие связи, если есть  
 ) {  
 return response()->json([  
 'message' => 'Нельзя удалить пункт выдачи, так как есть заказы, связанные с ним.'  
 ], 409);  
 }  
  
 $point->delete();  
  
 return response()->json(['message' => 'Пункт выдачи удалён.'], 200);  
 }  
}

***Листинг 11. Контроллер ProductController для продуктов***

class ProductController  
{  
 // Метод для просмотра всех товаров  
 public function index()  
 {  
 $products = Product::*all*();  
 if ($products->isEmpty()) {  
 throw new ApiException( 'Не найдено', 404);  
 }  
 return response()->json($products)->setStatusCode(200);  
 }  
 public function store(CreateProductRequest $request)  
 {  
 if (Auth::user()->role->code != 'admin' && Auth::user()->role->code != 'manager'){  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 // Обработка фото товара  
 $photoUrl = null;  
 if ($request->hasFile('photo') && $request->file('photo')->isValid()) {  
 $photoPath = $request->file('photo')->store('products', 'public');  
 $photoUrl = url('storage/' . $photoPath);  
 }  
  
 // Создаём товар с начальным quantity = 0  
 $product = Product::*create*([  
 'photo' => $photoUrl,  
 'name' => $request->name,  
 'description' => $request->description,  
 'sex' => $request->sex,  
 'quantity' => 0, // Устанавливаем начальное значение  
 'price' => $request->price,  
 'category\_id' => $request->category\_id,  
 'country\_id' => $request->country\_id,  
 ]);  
  
 $totalQuantity = 0;  
  
 // Обработка цветов и размеров  
 foreach ($request->colors as $colorData) {  
 // Обработка цвета (существующий или новый)  
 if (isset($colorData['color\_id'])) {  
 // Используем существующий цвет  
 $colorId = $colorData['color\_id'];  
 } else {  
 // Создаем новый цвет  
 $color = Color::*create*([  
 'name' => $colorData['new\_color\_name'],  
 'hex' => $colorData['new\_color\_hex'],  
 ]);  
 $colorId = $color->id;  
 }  
  
 // Обработка размеров для текущего цвета  
 foreach ($colorData['sizes'] as $sizeData) {  
 // Обработка размера (существующий или новый)  
 if (isset($sizeData['size\_id'])) {  
 // Используем существующий размер  
 $sizeId = $sizeData['size\_id'];  
 } else {  
 // Создаем новый размер  
 $size = Size::*create*([  
 'name' => $sizeData['new\_size\_name'],  
 ]);  
 $sizeId = $size->id;  
 }  
  
 // Создаем связь продукта с цветом и размером  
 ProductColorSize::*create*([  
 'product\_id' => $product->id,  
 'color\_id' => $colorId,  
 'size\_id' => $sizeId,  
 'quantity' => $sizeData['quantity'],  
 ]);  
  
 $totalQuantity += $sizeData['quantity'];  
 }  
 }  
  
 // Обновляем общее количество товара  
 $product->update(['quantity' => $totalQuantity]);  
  
 return response()->json([  
 'product' => $product,  
 'message' => 'Товар успешно создан с цветами и размерами'  
 ], 201);  
 }  
 // Метод для просмотра товара  
 public function show($id)  
 {  
 $product = Product::*with*(['category','country','productColorSizes.color', 'productColorSizes.size'])->find($id);  
  
 if (!$product) {  
 return response()->json(['message' => 'Товар не найден'], 404);  
 }  
  
 return response()->json($product, 200);  
 }  
 public function update(UpdateProductRequest $request, $id)  
 {  
 // Проверка прав доступа  
 if (Auth::user()->role->code != 'admin' && Auth::user()->role->code != 'manager') {  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 // Поиск товара с его вариантами  
 $product = Product::*with*(['productColorSizes.carts', 'productColorSizes.orderItems'])->find($id);  
 if (!$product) {  
 return response()->json(['message' => 'Товар не найден'], 404);  
 }  
  
 // Обработка основного фото  
 if ($request->hasFile('photo') && $request->file('photo')->isValid()) {  
 $photoPath = $request->file('photo')->store('products', 'public');  
 $photoUrl = url('storage/' . $photoPath);  
 $product->photo = $photoUrl;  
 }  
  
 $totalQuantity = 0;  
 $usedPcsIds = [];  
  
 // Обработка цветов и размеров  
 foreach ($request->colors as $colorData) {  
 $colorId = $this->getOrCreateColor($colorData);  
  
 foreach ($colorData['sizes'] as $sizeData) {  
 $sizeId = $this->getOrCreateSize($sizeData);  
  
 // Ищем или создаем связь товара с цветом и размером  
 $pcs = ProductColorSize::*firstOrNew*([  
 'product\_id' => $product->id,  
 'color\_id' => $colorId,  
 'size\_id' => $sizeId,  
 ]);  
  
 $pcs->quantity = $sizeData['quantity'];  
 $pcs->save();  
 $usedPcsIds[] = $pcs->id;  
 $totalQuantity += $sizeData['quantity'];  
 }  
 }  
  
 // Удаляем только те варианты, которых:  
 // 1. Нет в новом запросе  
 // 2. Нет в корзинах  
 // 3. Нет в заказах  
 $toDelete = $product->productColorSizes()  
 ->whereNotIn('id', $usedPcsIds)  
 ->whereDoesntHave('carts')  
 ->whereDoesntHave('orderItems')  
 ->get();  
  
 foreach ($toDelete as $pcs) {  
 $pcs->delete();  
 }  
  
 // Обновляем основные данные товара  
 $product->update([  
 'name' => $request->name,  
 'description' => $request->description,  
 'sex' => $request->sex,  
 'price' => $request->price,  
 'category\_id' => $request->category\_id,  
 'country\_id' => $request->country\_id,  
 'quantity' => $totalQuantity,  
 ]);  
  
 return response()->json([  
 'product' => $product->fresh()->load(['productColorSizes.color', 'productColorSizes.size']),  
 'message' => 'Товар успешно обновлен',  
 'warning' => $toDelete->count() < count($product->productColorSizes) - count($usedPcsIds)  
 ? 'Некоторые варианты не были удалены, так как они используются в заказах или корзинах'  
 : null  
 ], 200);  
 }  
  
 protected function getOrCreateColor($colorData)  
 {  
 if (isset($colorData['color\_id'])) {  
 return $colorData['color\_id'];  
 }  
  
 return Color::*create*([  
 'name' => $colorData['new\_color\_name'],  
 'hex' => $colorData['new\_color\_hex'],  
 ])->id;  
 }  
  
 protected function getOrCreateSize($sizeData)  
 {  
 if (isset($sizeData['size\_id'])) {  
 return $sizeData['size\_id'];  
 }  
  
 return Size::*create*([  
 'name' => $sizeData['new\_size\_name'],  
 ])->id;  
 }  
 public function destroy($id)  
 {  
 if (Auth::user()->role->code != 'admin' && Auth::user()->role->code != 'manager'){  
 return response()->json(['message' => 'У вас нет прав на выполнение этого действия'], 403);  
 }  
  
 $product = Product::*find*($id);  
  
 if (!$product) {  
 return response()->json(['message' => 'Товар не найден'], 404);  
 }  
  
 try {  
 // Поиск товара с учетом связанных данных  
 $product = Product::*with*('productColorSizes')->find($id);  
  
 if (!$product) {  
 return response()->json([  
 'message' => 'Товар не найден',  
 'success' => false  
 ], 404);  
 }  
  
 // Удаление связанных записей (цвета/размеры)  
 $product->productColorSizes()->delete();  
  
 // Удаление самого товара  
 $product->delete();  
  
 return response()->json([  
 'message' => 'Товар успешно удален',  
 'success' => true  
 ], 200);  
  
 } catch (\Illuminate\Database\QueryException $e) {  
 // Код ошибки для нарушения ограничения внешнего ключа  
 if ($e->getCode() == '23000') {  
 return response()->json([  
 'message' => 'Невозможно удалить товар: существуют связанные записи в корзине или других таблицах',  
 'success' => false  
 ], 409); // 409 Conflict - подходящий код статуса  
 }  
  
 // Другие ошибки БД  
 return response()->json([  
 'message' => 'Произошла ошибка при удалении товара',  
 'success' => false,  
 'error' => $e->getMessage()  
 ], 500);  
 }  
 }  
}

***Листинг 12. Контроллер ProfileController для профиля***

class ProfileController  
{  
 // Просмотр профиля  
 public function show()  
 {  
 $user = Auth::user()->load('role');  
  
 return response()->json([  
 'id' => $user->id,  
 'name' => $user->name,  
 'email' => $user->email,  
 'telephone' => $user->telephone,  
 'sex' => $user->sex,  
 'role' => $user->role->name ?? null,  
 'created\_at' => $user->created\_at,  
 'updated\_at' => $user->updated\_at,  
 ]);  
 }  
 // Обновление профиля  
 public function update(UpdateProfileRequest $request)  
 {  
 $user = Auth::user();  
  
 // Извлекаем валидированные данные  
 $validated = $request->validated();  
  
 $user->update($validated);  
  
 return response()->json([  
 'message' => 'Профиль успешно обновлен',  
 'data' => $user->fresh()->load('role')  
 ]);  
 }  
}

***Листинг 13. Контроллер ReviewController для отзывов***

class ReviewController  
{  
 public function index($id) {  
 $product = Product::*find*($id);  
 if (!$product) {  
 throw new ApiException('Товар не найден', 404);  
 }  
  
 $reviews = Review::*with*(['user' => function($query) {  
 $query->select('id', 'name');  
 }])  
 ->where('product\_id', $id)  
 ->orderBy('created\_at', 'desc')  
 ->get();  
  
 if ($reviews->isEmpty()) {  
 throw new ApiException('Отзывы для данного товара не указаны', 404);  
 }  
  
 return response()->json($reviews);  
 }  
 public function store(CreateReviewRequest $request, $id)  
 {  
 $user = Auth::user();  
  
 // Только обычные пользователи могут оставлять отзывы  
 if ($user->role->code != 'user') {  
 return response()->json(['message' => 'Только обычные пользователи могут оставлять отзывы'], 403);  
 }  
  
 // Проверка: уже есть отзыв на этот товар от пользователя  
 $existingReview = Review::*where*('user\_id', $user->id)  
 ->where('product\_id', $id)  
 ->first();  
  
 if ($existingReview) {  
 return response()->json(['message' => 'Вы уже оставили отзыв на этот товар'], 400);  
 }  
  
 // Проверка: есть ли у пользователя заказ с этим товаром и статусом id = 3  
 $hasCompletedOrder = Order::*where*('user\_id', $user->id)  
 ->where('status\_id', 3)  
 ->whereHas('orderItems.productColorSize', function ($query) use ($id) {  
 $query->where('product\_id', $id);  
 })  
 ->exists();  
  
 if (!$hasCompletedOrder) {  
 return response()->json(['message' => 'Вы можете оставить отзыв только после покупки и завершения заказа'], 403);  
 }  
  
 // Создаём отзыв  
 $review = Review::*create*([  
 'rating' => $request->rating,  
 'description' => $request->description,  
 'user\_id' => $user->id,  
 'product\_id' => $id,  
 ]);  
  
 return response()->json($review, 201);  
 }  
  
  
 public function destroy($productId, $reviewId)  
 {  
 $user = Auth::user();  
 $product = Product::*find*($productId);  
 $review = Review::*find*($reviewId);  
  
 // Проверка существования товара и отзыва  
 if (!$product) {  
 throw new ApiException('Товар не найден', 404);  
 }  
 if (!$review) {  
 throw new ApiException('Отзыв не найден', 404);  
 }  
  
 // Проверка соответствия отзыва и товара  
 if ($review->product\_id != $product->id) {  
 throw new ApiException('Этот отзыв не принадлежит указанному товару', 400);  
 }  
  
 // Проверка прав доступа  
 $isAdminOrManager = in\_array($user->role->code, ['admin', 'manager']);  
 $isReviewAuthor = $review->user\_id == $user->id;  
  
 if (!$isAdminOrManager && !$isReviewAuthor) {  
 return response()->json([  
 'message' => 'Недостаточно прав для удаления отзыва'  
 ], 403);  
 }  
  
 // Удаление отзыва  
 $review->delete();  
  
 return response()->json([  
 'message' => 'Отзыв успешно удален'  
 ], 200);  
 }  
}

***Листинг 14. Контроллер SizeController для размеров***

class SizeController  
{  
 public function index()  
 {  
 $sizes = Size::*all*();  
 if ($sizes->isEmpty()) {  
 throw new ApiException( 'Не найдено', 404);  
 }  
 return response()->json($sizes)->setStatusCode(200);  
 }  
}

***Листинг 15. Контроллер StatusController для статусов***

class StatusController  
{  
 public function index()  
 {  
 $statuses = Status::*all*();  
 if ($statuses->isEmpty()) {  
 throw new ApiException( 'Не найдено', 404);  
 }  
 return response()->json($statuses)->setStatusCode(200);  
 }  
}

***Листинг 16. Контроллер UserController для обычных пользователей и менеджеров***

public function createManager(RegisterRequest $request)  
 {  
 // Извлекаем role\_id для роли 'Менеджер' (предполагаем, что у менеджера code = 'manager')  
 $role\_id = Role::*where*('code', 'manager')->first()->id;  
  
 // Если роль не найдена, используем явное значение 2  
 if (!$role\_id) {  
 $role\_id = 2;  
 }  
  
 // Извлекаем валидированные данные  
 $validated = $request->validated();  
  
 // Создаем нового пользователя-менеджера  
 $user = User::*create*([  
 ...$validated,  
 'role\_id' => $role\_id  
 ]);  
  
 // Создание токена для пользователя  
 $user->api\_token = Hash::make(Str::*random*(60));  
 $user->save();  
  
 // Возвращаем ответ с токеном  
 return response()->json([  
 'user' => $user,  
 'token' => $user->api\_token,  
 'message' => 'Менеджер успешно создан'  
 ])->setStatusCode(201);  
 }  
 // Удаление менеджера (для админов и менеджеров)  
 public function deleteManager($id)  
 {  
 $currentUser = Auth::user();  
 $managerToDelete = User::*with*('role')->findOrFail($id);  
  
 // Проверяем, что удаляемый пользователь - менеджер  
 if ($managerToDelete->role->code !== 'manager') {  
 return response()->json(['message' => 'Можно удалять только менеджеров'], 400);  
 }  
  
 // Проверяем права текущего пользователя  
 if (!in\_array($currentUser->role->code, ['admin'])) {  
 return response()->json(['message' => 'Недостаточно прав для удаления менеджера'], 403);  
 }  
  
 $managerToDelete->delete();  
  
 return response()->json([  
 'message' => 'Менеджер успешно удален',  
 'deleted\_manager' => $this->formatManagerData($managerToDelete)  
 ]);  
 }  
  
 // Список всех менеджеров (для админов и менеджеров)  
 public function listManagers()  
 {  
 $currentUser = Auth::user();  
  
 if (!in\_array($currentUser->role->code, ['admin', 'manager'])) {  
 return response()->json(['message' => 'Доступ запрещен'], 403);  
 }  
  
 $managers = User::*whereHas*('role', function($query) {  
 $query->where('code', 'manager');  
 })->get()->map(function($manager) {  
 return $this->formatManagerData($manager);  
 });  
  
 return response()->json($managers);  
 }  
  
 // Форматирование данных менеджера  
 protected function formatManagerData($user)  
 {  
 return [  
 'id' => $user->id,  
 'name' => $user->name,  
 'email' => $user->email,  
 'telephone' => $user->telephone,  
 'role' => $user->role->name,  
 'created\_at' => $user->created\_at->format('Y-m-d H:i:s'),  
 ];  
 }  
 // Просмотр всех пользователей (для админа)  
 public function index()  
 {  
 if (Auth::user()->role->code !== 'admin') {  
 return response()->json(['message' => 'Доступ запрещен'], 403);  
 }  
  
 $users = User::*with*('role')->get()->map(function ($user) {  
 return $this->formatUserData($user);  
 });  
  
 return response()->json($users);  
 }  
  
 // Просмотр конкретного пользователя  
 public function show($id)  
 {  
 $currentUser = Auth::user();  
 $requestedUser = User::*with*('role')->findOrFail($id);  
  
 // Проверка прав: админ или просмотр своего профиля  
 if ($currentUser->role->code !== 'admin' && $currentUser->id != $id) {  
 return response()->json(['message' => 'Доступ запрещен'], 403);  
 }  
  
 return response()->json($this->formatUserData($requestedUser));  
 }  
  
 // Обновление пользователя  
 public function update(Request $request, $id)  
 {  
 $currentUser = Auth::user();  
 $user = User::*findOrFail*($id);  
  
 // Только админ может изменять других пользователей  
 if ($currentUser->role->code !== 'admin' && $currentUser->id != $id) {  
 return response()->json(['message' => 'Доступ запрещен'], 403);  
 }  
  
 // Валидация  
 $validated = $request->validate([  
 'name' => 'sometimes|string|max:255',  
 'email' => 'sometimes|email|unique:users,email,' . $id,  
 'telephone' => 'sometimes|string|max:20',  
 'sex' => 'sometimes|boolean',  
 'role\_id' => 'sometimes|integer|exists:roles,id'  
 ]);  
  
 // Только админ может менять роль  
 if (isset($validated['role\_id'])) {  
 if ($currentUser->role->code !== 'admin') {  
 return response()->json(['message' => 'Недостаточно прав для изменения роли'], 403);  
 }  
  
 // Запрещаем изменять роль последнему админу  
 if ($user->role->code === 'admin' &&  
 User::*where*('role\_id', $user->role\_id)->count() <= 1) {  
 return response()->json(['message' => 'Нельзя изменить роль последнему администратору'], 400);  
 }  
 }  
  
 $user->update($validated);  
  
 return response()->json([  
 'user' => $this->formatUserData($user->fresh('role')),  
 'message' => 'Данные пользователя обновлены'  
 ]);  
 }  
  
 // Форматирование данных пользователя  
 protected function formatUserData($user)  
 {  
 return [  
 'id' => $user->id,  
 'name' => $user->name,  
 'email' => $user->email,  
 'telephone' => $user->telephone,  
 'sex' => $user->sex,  
 'role' => [  
 'id' => $user->role->id,  
 'name' => $user->role->name,  
 'code' => $user->role->code  
 ],  
 'created\_at' => $user->created\_at,  
 'updated\_at' => $user->updated\_at,  
 ];  
 }  
}

***Листинг 17. Контроллер ViewFilterController для вывода категорий и стран***

class ViewFilterController  
{  
 public function allCountryIndex()  
 {  
 $countries = Country::*all*();  
 if ($countries->isEmpty()) {  
 throw new ApiException('Не найдено', 404);  
 }  
 return response()->json($countries)->setStatusCode(200);  
 }  
  
 public function allCategoryIndex()  
 {  
 $categories = Category::*all*();  
 if ($categories->isEmpty()) {  
 throw new ApiException( 'Не найдено', 404);  
 }  
 return response()->json($categories)->setStatusCode(200);  
 }  
}

***Листинг 18. Контроллер CartController для корзины***

class СartController  
{  
 // Получение корзины с товарами для пользователя  
 public function show(Request $request)  
 {  
 $clientId = $request->user()->id;  
  
 $cartProducts = Cart::*with*(['productColorSize.product', 'productColorSize.color', 'productColorSize.size'])  
 ->where('user\_id', $clientId)  
 ->get();  
  
 if ($cartProducts->isEmpty()) {  
 return response()->json(['message' => 'Корзина пуста'], 404);  
 }  
  
 $products = $cartProducts->map(function ($cartProduct) {  
 $pcs = $cartProduct->productColorSize;  
  
 return [  
 'id' => $pcs->id,  
 'name' => $pcs->product->name,  
 'price' => $pcs->product->price,  
 'photo' => $pcs->product->photo,  
 'description' => $pcs->product->description,  
 'color' => $pcs->color->name ?? null,  
 'size' => $pcs->size->name ?? null,  
 'quantity' => $cartProduct->quantity,  
 ];  
 });  
  
 $totalCost = $products->sum(function ($product) {  
 return $product['price'] \* $product['quantity'];  
 });  
  
 return response()->json([  
 'total\_cost' => $totalCost,  
 'created\_at' => $cartProducts->first()->created\_at,  
 'updated\_at' => $cartProducts->first()->updated\_at,  
 'products' => $products,  
 ]);  
 }  
  
 // Добавление товара в корзину  
 public function addProduct(Request $request, $productColorSizeId)  
 {  
 $clientId = $request->user()->id;  
 $quantity = $request->input('quantity', 1);  
  
 $pcs = ProductColorSize::*find*($productColorSizeId);  
  
 if (!$pcs) {  
 return response()->json(['message' => 'Товар не найден'], 404);  
 }  
  
 $existingCartProduct = Cart::*where*('user\_id', $clientId)  
 ->where('product\_color\_size\_id', $productColorSizeId)  
 ->first();  
  
 if ($existingCartProduct) {  
 $existingCartProduct->quantity += $quantity;  
  
 // Пересчёт total  
 $existingCartProduct->total = $pcs->product->price \* $existingCartProduct->quantity;  
  
 $existingCartProduct->save();  
 } else {  
 Cart::*create*([  
 'user\_id' => $clientId,  
 'product\_color\_size\_id' => $productColorSizeId,  
 'quantity' => $quantity,  
 'total' => $pcs->product->price \* $quantity,  
 ]);  
 }  
  
  
 return response()->json([  
 'status' => 'success',  
 'message' => 'Товар добавлен в корзину',  
 ]);  
 }  
  
 // Обновление количества товара в корзине  
 public function updateProduct(Request $request, $productColorSizeId)  
 {  
 $clientId = $request->user()->id;  
 $quantity = $request->input('quantity');  
  
 if ($quantity < 1) {  
 return response()->json([  
 'status' => 'error',  
 'message' => 'Количество товара должно быть больше 0'  
 ], 400);  
 }  
  
 $cartProduct = Cart::*where*('user\_id', $clientId)  
 ->where('product\_color\_size\_id', $productColorSizeId)  
 ->with('productColorSize.product')  
 ->first();  
  
 if (!$cartProduct) {  
 return response()->json([  
 'status' => 'error',  
 'message' => 'Товар не найден в корзине'  
 ], 404);  
 }  
  
 $cartProduct->quantity = $quantity;  
 $cartProduct->total = $cartProduct->productColorSize->product->price \* $quantity;  
 $cartProduct->save();  
  
 // Пересчёт всей корзины  
 $cartProducts = $this->getUserCartProducts($clientId);  
 $totalCost = $cartProducts->sum('total');  
  
 return response()->json([  
 'status' => 'success',  
 'message' => 'Количество товара в корзине успешно обновлено',  
 'data' => [  
 'total\_cost' => $totalCost  
 ]  
 ]);  
 }  
  
 // Удаление товара из корзины  
 public function removeProduct(Request $request, $productColorSizeId)  
 {  
 $clientId = $request->user()->id;  
  
 try {  
 // Проверяем существование товара в корзине пользователя  
 $cartItem = Cart::*where*('user\_id', $clientId)  
 ->where('product\_color\_size\_id', $productColorSizeId)  
 ->firstOrFail();  
  
 // Удаляем товар из корзины  
 $cartItem->delete();  
  
 // Получаем обновленную корзину  
 $cartProducts = $this->getUserCartProducts($clientId);  
 $totalCost = $cartProducts->sum('total');  
  
 return response()->json([  
 'status' => 'success',  
 'message' => 'Товар успешно удалён из корзины',  
 'data' => [  
 'total\_cost' => $totalCost,  
 'remaining\_items' => $cartProducts->count()  
 ]  
 ]);  
  
 } catch (\Illuminate\Database\Eloquent\ModelNotFoundException $e) {  
 return response()->json([  
 'status' => 'error',  
 'message' => 'Указанный товар не найден в вашей корзине'  
 ], 404);  
 } catch (\Exception $e) {  
 return response()->json([  
 'status' => 'error',  
 'message' => 'Произошла ошибка при удалении товара из корзины: ' . $e->getMessage()  
 ], 500);  
 }  
 }  
  
 // Вспомогательный метод для получения корзины пользователя  
 private function getUserCartProducts($clientId)  
 {  
 return Cart::*where*('user\_id', $clientId)->get();  
 }  
}

## Валидация запроса формы

В Laravel валидация запроса формы обычно выполняется в контроллере, используя встроенные инструменты валидации.

Form Request - это класс где мы описываем правила валидации для входящих данных. Обычно класс содержит набор правил под запрос из клиента. Мы можем его декларировать в контроллере, и через контейнер в Laravel он автоматически проверит данные на соответствии нашим правилам и через внутренние механизмы фреймворка выдаст ответ клиенту.

Для более сложных сценариев валидации вы можете создать «запрос формы». Запрос формы – это ваш класс запроса, который инкапсулирует свою собственную логику валидации и авторизации. Сгенерируем родительский запрос формы ApiRequest c методами непрошедшей авторизации и непрошедшей валидации.

Класс ApiRequest, можно увидеть на листинге 19, расширяет FormRequest и переопределяет два метода: failedValidation и failedAuthorization.

***Листинг 19. Класс ApiRequest c методами непрошедшей авторизации и непрошедшей валидации***

class ApiRequest extends FormRequest  
{  
 // Вызов исключения при провале валидации данных  
 public function failedValidation(Validator $validator)  
 {  
 throw new ApiException('Ошибка валидации данных', 422, $validator->errors());  
 }  
 // Вызов исключения при провале авторизации  
 public function failedAuthorization()  
 {  
 throw new ApiException('Ошибка доступа пользователя', 403);  
 }  
}

Класс RegisterRequest, виден на листинге 20, расширяет ApiRequest и определяет правила валидации для регистрации пользователя.

***Листинг 20. Класс RegisterRequest с правилами валидации для регистрации пользователя***

class RegisterRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'name' => 'required|string|max:64|min:3',  
 'email' => 'required|string|max:64|min:3|unique:users,email,',  
 'telephone' => 'nullable|string|max:25|min:10',  
 'sex' => 'required|boolean',  
 'password' => 'required|string|max:64|regex:/^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$/'  
 ];  
 }  
 public function messages(): array{  
 return [  
 'name.required' => 'Поле "Имя" обязательно для заполнения.',  
 'name.max' => 'Поле "Имя" не должно превышать 64 символа.',  
 'name.min' => 'Поле "Имя" не должно быть меньше 3 символов.',  
  
 'email.required' => 'Поле "Email" обязательно для заполнения.',  
 'email.max' => 'Поле "Email" не должно превышать 64 символа.',  
 'email.min' => 'Поле "Email" не должно быть меньше 3 символов.',  
 'email.unique' => 'Пользователь с вашей почтой уже зарегистрирован',  
  
 'telephone.max' => 'Поле "Телефон" не должно превышать 25 символов.',  
 'telephone.min' => 'Поле "Телефон" не должно быть меньше 3 символов.',  
  
 'sex.required' => 'Поле "Пол" обязательно для заполнения.',  
  
 'password.required' => 'Поле "Пароль" обязательно для заполнения.',  
 'password.max' => 'Поле "Пароль" не должно превышать 64 символа.',  
 'password.regex' => 'Пароль должен содержать как минимум одну заглавную букву, одну строчную букву, одну цифру и один специальный символ (@$!%\*?&).',  
 ];  
 }  
}

Класс FilterRequest, виден на листинге 21, расширяет ApiRequest и определяет правила валидации для фильтра поиска товаров по параметрам.

***Листинг 21. Класс FilterRequest с правилами валидации для фильтра поиска товаров по параметрам***

class FilterRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
  
 public function rules(): array  
 {  
 return [  
 'category\_id' => 'sometimes|integer|exists:categories,id',  
 'country\_id' => 'sometimes|integer|exists:countries,id',  
 'min\_price' => 'sometimes|numeric|min:0',  
 'max\_price' => 'sometimes|numeric|min:1|gt:min\_price',  
 'sex' => 'sometimes|boolean',  
 ];  
 }  
 public function messages(): array  
 {  
 return [  
 'exists' => 'Указанный :attribute не существует',  
 'gt' => 'Максимальная цена должна быть больше минимальной',  
  
 'category\_id.integer' => 'ID категории должно быть целым числом',  
 'country\_id.integer' => 'ID страны должно быть целым числом',  
  
 'min\_price.numeric' => 'Минимальная цена должна быть числом',  
 'max\_price.numeric' => 'Максимальная цена должна быть числом',  
 'min\_price.min' => 'Минимальная цена не может быть меньше 0',  
 'max\_price.min' => 'Максимальная цена не может быть меньше 1',  
  
 'sex.boolean' => 'Пол должен быть true или false, 1 или 0'  
 ];  
 }  
}

Класс CreatePointRequest, виден на листинге 22, расширяет ApiRequest и определяет правила валидации для создания пункта выдачи.

***Листинг 22. Класс CreatePointRequest с правилами валидации для создания пункта выдачи***

class CreatePointRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
  
 public function rules(): array  
 {  
 return [  
 'city' => 'required|string|max:100|min:2',  
 'street' => 'required|string|max:100|min:1',  
 'house' => 'required|string|max:10|min:1',  
 ];  
 }  
 public function messages(): array{  
 return [  
 'city.required' => 'Поле "Город" обязательно для заполнения.',  
 'city.max' => 'Поле "Город" не должно превышать 100 символов.',  
 'city.min' => 'Поле "Город" не должно быть меньше 2 символов.',  
  
 'street.required' => 'Поле "Улица" обязательно для заполнения.',  
 'street.max' => 'Поле "Улица" не должно превышать 100 символов.',  
 'street.min' => 'Поле "Улица" не должно быть меньше 1 символа.',  
  
 'house.required' => 'Поле "Дом" обязательно для заполнения.',  
 'house.max' => 'Поле "Дом" не должно превышать 25 символов.',  
 'house.min' => 'Поле "Дом" не должно быть меньше 1 символа.',  
 ];  
 }  
}

Класс UpdatePointRequest, виден на листинге 23, расширяет ApiRequest и определяет правила валидации для редактирования пункта выдачи.

***Листинг 23. Класс UpdatePointRequest с правилами валидации для редактирования пункта выдачи***

class UpdatePointRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
  
 public function rules(): array  
 {  
 return [  
 'city' => 'string|max:100|min:2',  
 'street' => 'string|max:100|min:1',  
 'house' => 'string|max:10|min:1',  
 ];  
 }  
 public function messages(): array{  
 return [  
 'city.max' => 'Поле "Город" не должно превышать 100 символов.',  
 'city.min' => 'Поле "Город" не должно быть меньше 2 символов.',  
  
 'street.max' => 'Поле "Улица" не должно превышать 100 символов.',  
 'street.min' => 'Поле "Улица" не должно быть меньше 1 символа.',  
  
 'house.max' => 'Поле "Дом" не должно превышать 25 символов.',  
 'house.min' => 'Поле "Дом" не должно быть меньше 1 символа.',  
 ];  
 }  
}

Класс CreateProductRequest, виден на листинге 24, расширяет ApiRequest и определяет правила валидации для создания товара.

***Листинг 24. Класс CreateProductRequest с правилами валидации для создания товара***

class CreateProductRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
  
 public function rules(): array  
 {  
 return [  
 'photo' => 'nullable|file|image|max:2048',  
 'name' => 'required|string|max:255',  
 'description' => 'required|string',  
 'sex' => 'required|boolean',  
 'price' => 'required|numeric|min:0',  
 'category\_id' => 'required|integer|exists:categories,id',  
 'country\_id' => 'required|integer|exists:countries,id',  
 'colors' => 'required|array',  
 'colors.\*.color\_id' => 'nullable|exists:colors,id',  
 'colors.\*.new\_color\_name' => 'nullable|required\_with:colors.\*.new\_color\_hex|string|max:255',  
 'colors.\*.new\_color\_hex' => 'nullable|required\_with:colors.\*.new\_color\_name|string|regex:/^#[a-fA-F0-9]{6}$/',  
 'colors.\*.sizes' => 'required|array',  
 'colors.\*.sizes.\*.size\_id' => 'nullable|exists:sizes,id',  
 'colors.\*.sizes.\*.new\_size\_name' => 'nullable|string|max:255',  
 'colors.\*.sizes.\*.quantity' => 'required|integer|min:0',  
 ];  
 }  
 public function messages(): array{  
 return [  
 'name.required' => 'Поле "Название" обязательно для заполнения.',  
 'name.max' => 'Поле "Название" не должно превышать 255 символов.',  
  
 'description.required' => 'Поле "Описание" обязательно для заполнения.',  
  
 'sex.required' => 'Поле "Пол" обязательно для заполнения.',  
  
 'price.required' => 'Поле "Цена" обязательно для заполнения.',  
 'price.min' => 'Поле "Цена" не может быть ниже 0.',  
  
 'category\_id' => 'Категория должна быть выбрана',  
 'category\_id.exists' => 'Выбранной категории не существует',  
 'country\_id' => 'Страна производителя должна быть выбрана',  
 'country\_id.exists' => 'Выбранной страны не существует',  
  
 'colors.required' => 'Необходимо указать хотя бы один цвет.',  
 'colors.min' => 'Необходимо указать хотя бы один цвет.',  
 'colors.max' => 'Нельзя указать более 20 цветов.',  
  
 'colors.\*.color\_id.exists' => 'Выбранный цвет не существует или был удален.',  
 'colors.\*.color\_id.required\_without\_all' => 'Укажите существующий цвет или создайте новый.',  
  
 'colors.\*.new\_color\_name.string' => 'Название нового цвета должно быть строкой.',  
 'colors.\*.new\_color\_name.max' => 'Название нового цвета не должно превышать 100 символов.',  
 'colors.\*.new\_color\_name.required\_with' => 'Название цвета обязательно при указании HEX-кода.',  
 'colors.\*.new\_color\_name.required\_without' => 'Укажите название нового цвета или выберите существующий.',  
  
 'colors.\*.new\_color\_hex.regex' => 'HEX-код цвета должен быть в формате #FFFFFF.',  
 'colors.\*.new\_color\_hex.required\_with' => 'HEX-код цвета обязателен при указании названия.',  
 'colors.\*.new\_color\_hex.required\_without' => 'Укажите HEX-код нового цвета или выберите существующий.',  
  
 'colors.\*.sizes.required' => 'Для каждого цвета укажите хотя бы один размер.',  
 'colors.\*.sizes.min' => 'Для каждого цвета укажите хотя бы один размер.',  
 'colors.\*.sizes.max' => 'Для каждого цвета нельзя указать более 20 размеров.',  
  
 'colors.\*.sizes.\*.size\_id.exists' => 'Выбранный размер не существует или был удален.',  
 'colors.\*.sizes.\*.size\_id.required\_without' => 'Укажите существующий размер или создайте новый.',  
  
 'colors.\*.sizes.\*.new\_size\_name.string' => 'Название нового размера должно быть строкой.',  
 'colors.\*.sizes.\*.new\_size\_name.max' => 'Название нового размера не должно превышать 50 символов.',  
 'colors.\*.sizes.\*.new\_size\_name.required\_without' => 'Укажите название нового размера или выберите существующий.',  
  
 'colors.\*.sizes.\*.quantity.required' => 'Укажите количество товара для размера.',  
 'colors.\*.sizes.\*.quantity.integer' => 'Количество должно быть целым числом.',  
 'colors.\*.sizes.\*.quantity.min' => 'Количество не может быть отрицательным.',  
 'colors.\*.sizes.\*.quantity.max' => 'Количество не может превышать 10000.',  
 ];  
 }  
}

Класс UpdateProductRequest, виден на листинге 25, расширяет ApiRequest и определяет правила валидации для редактирования товара.

***Листинг 25. Класс UpdateProductRequest с правилами валидации для редактирования товара***

class UpdateProductRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
  
 public function rules(): array  
 {  
 return [  
 'photo' => 'nullable|file|image|max:2048',  
 'name' => 'string|max:255',  
 'description' => 'string',  
 'sex' => 'boolean',  
 'price' => 'numeric|min:0',  
 'category\_id' => 'integer|exists:categories,id',  
 'country\_id' => 'integer|exists:countries,id',  
 'colors' => 'nullable|array',  
 'colors.\*.color\_id' => 'nullable|integer|exists:colors,id',  
 'colors.\*.sizes' => 'nullable|array',  
 'colors.\*.sizes.\*.size\_id' => 'nullable|integer|exists:sizes,id',  
 'colors.\*.sizes.\*.quantity' => 'nullable|integer|min:0',  
 ];  
 }  
 public function messages(): array{  
 return [  
 'name.max' => 'Поле "Название" не должно превышать 255 символов.',  
  
 'price.min' => 'Поле "Цена" не должно быть ниже 0.',  
  
 'category\_id' => 'Категория должна быть выбрана',  
 'category\_id.exists' => 'Выбранной категории не существует',  
 'country\_id' => 'Страна производителя должна быть выбрана',  
 'country\_id.exists' => 'Выбранной страны не существует',  
  
 'colors.required' => 'Необходимо указать хотя бы один цвет.',  
 'colors.min' => 'Необходимо указать хотя бы один цвет.',  
 'colors.max' => 'Нельзя указать более 20 цветов.',  
  
 'colors.\*.color\_id.exists' => 'Выбранный цвет не существует или был удален.',  
 'colors.\*.color\_id.required\_without\_all' => 'Укажите существующий цвет или создайте новый.',  
  
 'colors.\*.new\_color\_name.string' => 'Название нового цвета должно быть строкой.',  
 'colors.\*.new\_color\_name.max' => 'Название нового цвета не должно превышать 100 символов.',  
 'colors.\*.new\_color\_name.required\_with' => 'Название цвета обязательно при указании HEX-кода.',  
 'colors.\*.new\_color\_name.required\_without' => 'Укажите название нового цвета или выберите существующий.',  
  
 'colors.\*.new\_color\_hex.regex' => 'HEX-код цвета должен быть в формате #FFFFFF.',  
 'colors.\*.new\_color\_hex.required\_with' => 'HEX-код цвета обязателен при указании названия.',  
 'colors.\*.new\_color\_hex.required\_without' => 'Укажите HEX-код нового цвета или выберите существующий.',  
  
 'colors.\*.sizes.required' => 'Для каждого цвета укажите хотя бы один размер.',  
 'colors.\*.sizes.min' => 'Для каждого цвета укажите хотя бы один размер.',  
 'colors.\*.sizes.max' => 'Для каждого цвета нельзя указать более 20 размеров.',  
  
 'colors.\*.sizes.\*.size\_id.exists' => 'Выбранный размер не существует или был удален.',  
 'colors.\*.sizes.\*.size\_id.required\_without' => 'Укажите существующий размер или создайте новый.',  
  
 'colors.\*.sizes.\*.new\_size\_name.string' => 'Название нового размера должно быть строкой.',  
 'colors.\*.sizes.\*.new\_size\_name.max' => 'Название нового размера не должно превышать 50 символов.',  
 'colors.\*.sizes.\*.new\_size\_name.required\_without' => 'Укажите название нового размера или выберите существующий.',  
  
 'colors.\*.sizes.\*.quantity.required' => 'Укажите количество товара для размера.',  
 'colors.\*.sizes.\*.quantity.integer' => 'Количество должно быть целым числом.',  
 'colors.\*.sizes.\*.quantity.min' => 'Количество не может быть отрицательным.',  
 'colors.\*.sizes.\*.quantity.max' => 'Количество не может превышать 10000.',  
 ];  
 }  
}

Класс UpdateProfileRequest, виден на листинге 26, расширяет ApiRequest и определяет правила валидации для редактирования профиля.

***Листинг 26. Класс UpdateProfileRequest с правилами валидации для редактирования профиля***

class UpdateProfileRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
  
 public function rules(): array  
 {  
 return [  
 'name' => 'string|max:64|min:3',  
 'email' => 'string|max:64|min:3|unique:users,email,' . $this->user()->id,  
 'telephone' => 'string|max:25|min:10',  
 'sex' => 'boolean',  
 'password' => 'string|max:64|regex:/^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$/',  
 ];  
 }  
 public function messages(): array{  
 return [  
 'name.max' => 'Поле "Имя" не должно превышать 64 символа.',  
 'name.min' => 'Поле "Имя" не должно быть меньше 3 символов.',  
  
 'email.max' => 'Поле "Email" не должно превышать 64 символа.',  
 'email.min' => 'Поле "Email" не должно быть меньше 3 символов.',  
 'email.unique' => 'Пользователь с вашей почтой уже существует',  
  
 'telephone.max' => 'Поле "Телефон" не должно превышать 25 символов.',  
 'telephone.min' => 'Поле "Телефон" не должно быть меньше 3 символов.',  
  
 'sex.required' => 'Поле "Пол" обязательно для заполнения.',  
  
 'password.max' => 'Поле "Пароль" не должно превышать 64 символа.',  
 'password.regex' => 'Пароль должен содержать как минимум одну заглавную букву, одну строчную букву, одну цифру и один специальный символ (@$!%\*?&).',  
 ];  
 }  
}

Класс CreateReviewRequest, виден на листинге 27, расширяет ApiRequest и определяет правила валидации для создания отзыва.

***Листинг 27. Класс CreateReviewRequest с правилами валидации для создания отзыва***

class CreateReviewRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
  
 public function rules(): array  
 {  
 return [  
 'rating' => 'required|integer|max:5|min:1',  
 'description' => 'required|string|max:255',  
 ];  
 }  
 public function messages(): array{  
 return [  
 'rating.required' => 'Поле "Рейтинг" обязательно для заполнения.',  
 'rating.max' => 'Поле "Рейтинг" не должно превышать 5.',  
 'rating.min' => 'Поле "Рейтинг" не должно быть меньше 1.',  
  
 'description.required' => 'Поле "Описание" обязательно для заполнения.',  
 'description.max' => 'Поле "Описание" не должно превышать 255 символов.',  
 ];  
 }  
}

Класс UpdateReviewRequest, виден на листинге 28, расширяет ApiRequest и определяет правила валидации для редактирования отзыва.

***Листинг 28. Класс UpdateReviewRequest с правилами валидации для редактирования отзыва***

class UpdateReviewRequest extends ApiRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
  
 public function rules(): array  
 {  
 return [  
 'rating' => 'integer|max:5|min:1',  
 'description' => 'string|max:255',  
 ];  
 }  
 public function messages(): array{  
 return [  
 'rating.max' => 'Поле "Рейтинг" не должно превышать 5.',  
 'rating.min' => 'Поле "Рейтинг" не должно быть меньше 1.',  
  
 'description.max' => 'Поле "Описание" не должно превышать 255 символов.',  
 ];  
 }  
}

## Маршрутизация

Маршрутизация в Laravel - это процесс определения того, как приложение должно отвечать на различные HTTP-запросы. В фреймворке Laravel маршруты определяются в файле routes/web.php для веб-маршрутов и routes/api.php для API-маршрутов.

Маршруты, определенные в файле routes/api.php, вложены в группу маршрутов в RouteServiceProvider. Внутри этой группы автоматически применяется префикс URI/api, поэтому не нужно вручную добавлять его к каждому маршруту в файле маршрутов.

Исходя из технического задания были разработаны маршруты, которые записаны в файле routes\api.php и можно увидеть на листинге 29:

***Листинг 29. Маршрутизация***

// Регистрация пользователя  
Route::post('/register', [AuthController::class, 'register']);  
// Авторизация  
Route::post('/login', [AuthController::class, 'login']);  
// Выход  
Route::post('/logout', [AuthController::class, 'logout'])->middleware('auth:api');  
  
// Товары  
Route::get('/product', [ProductController::class, 'index']); // Просмотр всех товаров  
Route::post('/product', [ProductController::class, 'store'])->middleware('auth:api'); // Создание товара  
Route::get('/product/{id}', [ProductController::class, 'show']); // Просмотр конкретного товара  
Route::patch('/product/{id}', [ProductController::class, 'update'])->middleware('auth:api'); // Обновление товара  
Route::delete('/product/{id}', [ProductController::class, 'destroy'])->middleware('auth:api'); // Удаление товара  
  
// Профиль  
Route::get('/profile', [ProfileController::class, 'show'])->middleware('auth:api'); // Просмотр профиля  
Route::patch('/profile', [ProfileController::class, 'update'])->middleware('auth:api'); // Обновление своего профиля  
  
// Товары по фильтрам  
Route::get('/products/filter', [FilterController::class, 'filterProducts']); // Фильтр по категории, стране, цене и полу  
Route::get('/products/search', [FilterController::class, 'search']); // Поиск по тексту  
  
// Вывод категорий и стран  
Route::get('/country', [ViewFilterController::class, 'allCountryIndex']);  
Route::get('/category', [ViewFilterController::class, 'allCategoryIndex']);  
  
// Создание категории  
Route::post('/category', [CategoryController::class, 'store'])->middleware('auth:api');  
  
// Отзывы  
Route::get('/product/{id}/review', [ReviewController::class, 'index']); // Просмотр отзывов по товару  
Route::post('/product/{id}/review', [ReviewController::class, 'store'])->middleware('auth:api'); // Создание отзыва  
Route::delete('/product/{productId}/review/{reviewId}', [ReviewController::class, 'destroy'])->middleware('auth:api'); // Удаление отзыва  
  
// Пункты выдачи  
Route::get('/point', [PointController::class, 'index']); // Просмотр всех пунктов выдачи  
Route::post('/point', [PointController::class, 'store'])->middleware('auth:api'); // Создание пункта выдачи  
Route::patch('/point/{id}', [PointController::class, 'update'])->middleware('auth:api'); // Редактирование пункта выдачи  
Route::delete('/point/{id}', [PointController::class, 'destroy'])->middleware('auth:api'); // Удаление пункта выдачи  
  
// Корзина  
Route::get('/cart', [СartController::class, 'show'])->middleware('auth:api'); // Просмотр товаров всех товаров в своей корзине  
Route::post('/cart/product/{productColorSizeId}', [СartController::class, 'addProduct'])->middleware('auth:api'); // Добаление товара в корзину  
Route::patch('/cart/product/{productColorSizeId}', [СartController::class, 'updateProduct'])->middleware('auth:api'); // Обновление количества товара в корзине  
Route::delete('/cart/product/{productColorSizeId}', [СartController::class, 'removeProduct'])->middleware('auth:api'); // Удаление товара из корзины  
  
// Заказы  
Route::get('/order', [OrderController::class, 'index'])->middleware('auth:api'); // Просмотр всех заказов или только своих (в зависимости от роли)  
Route::post('/order', [OrderController::class, 'store'])->middleware('auth:api'); // Создание заказа  
Route::get('/order/{id}', [OrderController::class, 'show'])->middleware('auth:api'); // Просмотр конкретного заказа  
Route::patch('/order/{id}', [OrderController::class, 'update'])->middleware('auth:api'); // Обновление статуса заказа  
Route::patch('/order/cancelled/{id}', [OrderController::class, 'destroy'])->middleware('auth:api'); // Отмена заказа  
Route::get('/checkout/success', [OrderController::class, 'handleSuccess'])->name('checkout.success'); // Оплачено  
Route::get('/checkout/cancel', [OrderController::class, 'handleCancel'])->name('checkout.cancel'); // Отменён  
  
// Менеджеры  
Route::post('/manager', [UserController::class, 'createManager'])->middleware('auth:api'); // Создание менеджера  
Route::delete('/manager/{id}', [UserController::class, 'deleteManager'])->middleware('auth:api'); // Удаление менеджера  
Route::get('/manager', [UserController::class, 'listManagers'])->middleware('auth:api'); // Список менеджеров  
  
// Пользователи  
Route::get('/user', [UserController::class, 'index'])->middleware('auth:api');  
Route::get('/user/{id}', [UserController::class, 'show'])->middleware('auth:api');  
Route::patch('/user/{id}', [UserController::class, 'update'])->middleware('auth:api');  
  
// Цвета и размеры  
Route::get('/colors', [ColorController::class, 'index']);  
Route::get('/sizes', [SizeController::class, 'index']);  
  
// Статусы  
Route::get('/statuses', [StatusController::class, 'index']);

# ГЛАВА 4. АВТОТЕСТИРОВАНИЕ API

В рамках разработки системы было проведено комплексное автоматизированное тестирование API с применением Postman. Были разработаны тестовые сценарии, охватывающие основные эндпоинты и сценарии взаимодействия с сервером.

## 401 Unauthorized

Код состояния HTTP 401 Unauthorized указывает на то, что запрос к API не может быть выполнен, поскольку отсутствует действующая аутентификация; это означает, что клиент должен предоставить учетные данные для доступа к защищенному ресурсу, например, предоставить верный токен в заголовке Authorization, либо у текущего пользователя недостаточно прав для доступа.

pm.test ("Проверка кода ответа", function () {

pm.response.to.have.status(401);

});

pm.test ("Проверка сообщения", function () {

pm.expect(pm.response.json()).to.have.property("message");

});

|  |  |
| --- | --- |
| **URL** | **МЕТОД** |
| http://{path}/api/login | POST |
| http://{path}/api/logout | POST |
| http://{path}/api/product | POST |
| http://{path}/api/product/{id} | PATCH |
| DELETE |
| http://{path}/api/profile | GET |
| PATCH |
| http://{path}/api/category | POST |
| http://{path}/api/product/{id}/review | POST |
| http://{path}/api/product/{productId}/review/{reviewId} | DELETE |
| http://{path}/api/point | POST |
| http://{path}/api/point/{id} | PATCH |
| DELETE |
| http://{path}/api/cart | GET |
| http://{path}/api/cart/{productColorSizeId} | POST |
| PATCH |
| DELETE |
| http://{path}/api/order | GET |
| POST |
| http://{path}/api/order/{id} | GET |
| PATCH |
| http://{path}/api/order/cancelled/{id} | PATCH |
| http://{path}/api/accesses-partial-cancel/{id} | POST |
| http://{path}/api/manager | POST |
| GET |
| http://{path}/api/manager/{id} | DELETE |
| http://{path}/api/user | GET |
| http://{path}/api/user/{id} | GET |
| PATCH |

## [403 Forbidden](https://ru.wikipedia.org/wiki/%D0%9E%D1%88%D0%B8%D0%B1%D0%BA%D0%B0_403)

Код состояния HTTP 403 Forbidden указывает на то, что сервер понимает запрос, но отказывается его выполнять, даже если клиент аутентифицирован; это означает, что у пользователя, даже после успешной аутентификации, недостаточно прав доступа к запрошенному ресурсу или эндпоинту, либо доступ к этому ресурсу запрещен по другим причинам, не связанным с аутентификацией.

pm.test ("Проверка кода ответа", function () {

pm.response.to.have.status(403);

});

pm.test ("Проверка сообщения", function () {

pm.expect(pm.response.json()).to.have.property("message");

});

|  |  |
| --- | --- |
| **URL** | **МЕТОД** |
| http://{path}/api/product | POST |
| http://{path}/api/product/{id} | PATCH |
| DELETE |
| http://{path}/api/category | POST |
| http://{path}/api/product/{id}/review | POST |
| http://{path}/api/point | POST |
| http://{path}/api/point/{id} | PATCH |
| http://{path}/api/point/{id} | DELETE |
| http://{path}/api/order/{id} | PATCH |
| http://{path}/api/manager | POST |
| GET |
| http://{path}/api/manager/{id} | DELETE |
| http://{path}/api/user | GET |
| http://{path}/api/user/{id} | GET |
| PATCH |

## 404 Not Found

Код состояния HTTP 404 Not Found указывает на то, что сервер не смог найти ресурс, соответствующий запрошенному URL; это означает, что запрошенный эндпоинт или ресурс не существует на сервере, либо был удален, либо URL был введен с ошибкой, и сервер не может обработать запрос, так как целевой ресурс не идентифицирован.

pm.test ("Проверка кода ответа", function () {

pm.response.to.have.status(404);

});

pm.test ("Проверка сообщения", function () {

pm.expect(pm.response.json()).to.have.property("message");

});

|  |  |
| --- | --- |
| **URL** | **МЕТОД** |
| http://{path}/api/product | GET |
| http://{path}/api/product/{id} | GET |
| PATCH |
| DELETE |
| http://{path}/api/products/filter | GET |
| http://{path}/api/products/search | GET |
| http://{path}/api/country | GET |
| http://{path}/api/category | GET |
| http://{path}/api/product/{id}/review | GET |
| http://{path}/api/product/{productId}/review/{reviewId} | DELETE |
| http://{path}/api/point | GET |
| http://{path}/api/point/{id} | PATCH |
| DELETE |
| http://{path}/api/cart | GET |
| http://{path}/api/cart/{productColorSizeId} | POST |
| PATCH |
| DELETE |
| http://{path}/api/order | GET |
| http://{path}/api/order/{id} | GET |
| PATCH |
| http://{path}/api/order/cancelled/{id} | PATCH |
| http://{path}/api/colors | GET |
| http://{path}/api/sizes | GET |
| http://{path}/api/statuses | GET |

## 200 OK

Код состояния HTTP 200 OK указывает на то, что запрос был успешно обработан сервером; это означает, что запрошенное действие было выполнено, и сервер возвращает запрошенные данные или подтверждение успешного выполнения операции в теле ответа.

pm.test ("Проверка кода ответа", function () {

pm.response.to.have.status(200);

});

pm.test ("Проверка сообщения", function () {

pm.expect(pm.response.json()).to.have.property("message");

});

|  |  |
| --- | --- |
| **URL** | **МЕТОД** |
| http://{path}/api/login | POST |
| http://{path}/api/logout | POST |
| http://{path}/api/product | GET |
| http://{path}/api/product/{id} | GET |
| PATCH |
| DELETE |
| http://{path}/api/profile | GET |
| PATCH |
| http://{path}/api/products/filter | GET |
| http://{path}/api/products/search | GET |
| http://{path}/api/country | GET |
| http://{path}/api/category | GET |
| http://{path}/api/product/{id}/review | GET |
| http://{path}/api/product/{productId}/review/{reviewId} | DELETE |
| http://{path}/api/point | GET |
| http://{path}/api/point/{id} | PATCH |
| DELETE |
| http://{path}/api/cart | GET |
| http://{path}/api/cart/{productColorSizeId} | POST |
| PATCH |
| DELETE |
| http://{path}/api/order | GET |
| POST |
| http://{path}/api/order/{id} | GET |
| PATCH |
| http://{path}/api/order/cancelled/{id} | PATCH |
| http://{path}/api/manager/{id} | DELETE |
| http://{path}/api/manager | GET |
| http://{path}/api/user | GET |
| http://{path}/api/user/{id} | GET |
| PATCH |
| http://{path}/api/colors | GET |
| http://{path}/api/sizes | GET |
| http://{path}/api/statuses | GET |

## 201 Created

Код состояния HTTP 201 Created указывает на то, что запрос был успешно обработан и в результате на сервере был создан новый ресурс; это означает, что сервер успешно создал новый элемент (например, запись в базе данных) на основе данных, предоставленных в запросе, и возвращает URI нового ресурса в заголовке Location.

pm.test ("Проверка кода ответа", function () {

pm.response.to.have.status(201);

});

pm.test ("Проверка сообщения", function () {

pm.expect(pm.response.json()).to.have.property("message");

});

|  |  |
| --- | --- |
| **URL** | **МЕТОД** |
| http://{path}/api/register | POST |
| http://{path}/api/product | POST |
| http://{path}/api/category | POST |
| http://{path}/api/product/{id}/review | POST |
| http://{path}/api/point | POST |
| http://{path}/api/manager | POST |

# ГЛАВА 5. РЕАЛИЗАЦИЯ ФУНКЦИОНАЛА ВЕБ-ПРИЛОЖЕНИЯ

Перед стартом разработки веб-приложения интернет-магазина "Тайны Востока" была проведена подготовка программной среды и настроены основные инструменты разработки.

В качестве базовой технологии был выбран JavaScript-фреймворк Vue 3, отличающийся гибкостью, производительностью и удобством для создания современных интерфейсов. Данный инструмент позволяет быстро реализовывать компоненты и эффективно управлять состоянием приложения.

Для установки необходимых библиотек и модулей применялся менеджер пакетов npm, благодаря которому проект получил все зависимости, сгруппированные в папке node\_modules.

В процессе работы использовался инструмент Vite, обеспечивающий мгновенный запуск проекта, поддержку модульной архитектуры и функцию горячей перезагрузки (Hot Module Replacement), что значительно ускоряет цикл разработки и тестирования.

Выбор этих технологий позволил создать стабильную и легко масштабируемую архитектуру клиентской части интернет-магазина.

## Реализация API и вспомогательных модулей

Для обеспечения взаимодействия клиентской части интернет-магазина "Тайны Востока" с серверной логикой на Laravel была реализована система API-запросов с использованием библиотеки Axios, которая широко применяется в разработке Vue-приложений для работы с HTTP-запросами.

Создание отдельного модуля для настройки API позволило централизовать конфигурацию всех запросов, стандартизировать их выполнение, а также упростить масштабирование и сопровождение проекта. Все обращения к серверу проходят через единый экземпляр Axios, что делает код чище и управляемее.

Модуль настройки API-запросов (api.js), можно увидеть на листинге 30.

В модуле api.js создаётся настраиваемый экземпляр Axios, в который включены следующие параметры:

* базовый URL, указывающий на backend (http://secrets-of-the-east.ru/api);
* заголовок по умолчанию 'Content-Type': 'application/json' для корректной передачи данных в формате JSON;
* перехватчик запроса (interceptor), автоматически добавляющий токен авторизации Bearer из localStorage ко всем исходящим запросам, если пользователь авторизован.

Такой подход обеспечивает безопасную авторизацию на стороне клиента и упрощает доступ к защищённым маршрутам API Laravel.

**Листинг 30. Модуль api.js**

import axios from 'axios';

const api = axios.create({

baseURL: 'http://secrets-of-the-east.ru/api',

headers: {

'Content-Type': 'application/json',

},

});

// Interceptor: добавляет токен к каждому запросу

api.interceptors.request.use(config => {

const token = localStorage.getItem('auth\_token');

if (token) {

config.headers.Authorization = `Bearer ${token}`;

}

return config;

}, error => {

return Promise.reject(error);

});

export default api;

## Реализация управления состоянием (Vuex Store)

Для централизованного управления данными на клиентской части интернет-магазина использовалась библиотека Vuex. Она обеспечивает глобальное хранилище, к которому имеют доступ все компоненты, и реализует реактивный подход к работе с данными. Такой подход упрощает взаимодействие между модулями приложения, особенно при наличии авторизации, фильтрации и отображения пользовательских данных.

Vuex используется для хранения информации о текущем пользователе, состоянии загрузки, а также взаимодействия с локальным хранилищем браузера (localStorage), что необходимо для сохранения авторизации между сессиями.

Хранилище было реализовано в файле store/index.js, можно увидеть на листинге 31. Оно включает четыре ключевых раздела: state, mutations, actions и getters.

**State — хранение состояния**

State — это централизованное хранилище всех данных, нужных для работы и отображения интерфейса приложения. В нашем проекте сюда попадают такие данные, как статус загрузки, сведения о текущем пользователе, его имя, email и роль. При старте приложения данные пользователя извлекаются из localStorage, чтобы сохранить сессию между перезагрузками страницы и обеспечить плавный пользовательский опыт.

Такой подход помогает избежать рассогласования данных и дублирования, даёт единый источник правды, что особенно важно для больших приложений.

**Листинг 31. Состояние (state) Vuex-хранилища**

state: {

isLoading: false, // Флаг для отображения состояния загрузки

user: {

loggedIn: !!localStorage.getItem('auth\_token'), // Проверка авторизации по наличию токена

name: localStorage.getItem('userName') || '', // Имя пользователя из localStorage

email: localStorage.getItem('userEmail') || '', // Email пользователя из localStorage

role: Number(localStorage.getItem('userRole')) || null, // Роль пользователя (1-админ, 2-менеджер, 3-пользователь)

},

**Mutations — синхронное изменение состояния**

Mutations обеспечивают единственный способ изменить состояние приложения. Они работают синхронно, что упрощает отладку и делает поведение приложения более предсказуемым. Все изменения state проходят через вызов мутаций, можно увидеть на листинге 32.

В проекте мутации реализуют установку состояния загрузки, сохранение данных пользователя в state и localStorage, а также очистку данных при выходе из системы. Это обеспечивает согласованность данных и их сохранность.

**Листинг 32. Мутации Vuex-хранилища**

mutations: {

setLoading(state, value) {

state.isLoading = value; // Устанавливаем флаг загрузки

},

setUser(state, user) {

// Обновляем состояние пользователя

state.user = {

loggedIn: true,

name: user.name,

email: user.email,

role: Number(user.role),

};

// Сохраняем данные в localStorage

localStorage.setItem('auth\_token', user.token);

localStorage.setItem('userName', user.name);

localStorage.setItem('userEmail', user.email);

localStorage.setItem('userRole', user.role);

},

logout(state) {

// Сбрасываем состояние пользователя

state.user = { loggedIn: false, name: '', email: '', role: null };

// Удаляем данные из localStorage

localStorage.removeItem('auth\_token');

localStorage.removeItem('userName');

localStorage.removeItem('userEmail');

localStorage.removeItem('userRole');

},

**Actions — асинхронная работа с API**

Actions позволяют выполнять асинхронные операции, например, запросы к серверу, и по окончании вызывать мутации для обновления состояния, можно увидеть на листинге 33. Это поддерживает однонаправленный поток данных и разделение логики.

В нашем приложении actions реализуют процесс логина (отправку данных на backend, получение токена и данных пользователя), выход из системы и управление индикатором загрузки. Обработка ошибок внутри actions повышает надёжность и информативность.

**Листинг 33. Действия Vuex-хранилища**

actions: {

async login({ commit }, userCredentials) {

try {

// Отправляем запрос на авторизацию

const response = await api.post('/login', userCredentials);

// Проверка структуры ответа

if (!response.data || !response.data.token) {

throw new Error('Неверная структура ответа сервера');

}

// Сохраняем данные пользователя

commit('setUser', {

token: response.data.token,

name: response.data.user.name || userCredentials.email,

email: response.data.user.email || userCredentials.email,

role: response.data.user.role\_id,

});

} catch (error) {

console.error('Ошибка авторизации:', error); // Логируем ошибку

throw error; // Прокидываем исключение

}

},

async logout({ commit }) {

try {

// Отправляем запрос на выход

await api.post('/logout');

} catch (error) {

console.warn('Ошибка при выходе с сервера:', error); // Логируем, но не блокируем выход

} finally {

commit('logout'); // Очищаем состояние

}

},

startLoading({ commit }) {

commit('setLoading', true); // Включаем индикатор загрузки

},

stopLoading({ commit }) {

commit('setLoading', false); // Выключаем индикатор загрузки

},

**Getters — получение производных данных**

Getters позволяют получать вычисленные или отфильтрованные данные из state без его изменения. Они выступают прослойкой между состоянием и компонентами, предоставляя удобный доступ к производным значениям, можно увидеть на листинге 34.

В нашем случае getters возвращают, например, статус авторизации пользователя, имя, роль, а также логические проверки типа — админ, менеджер или пользователь. Это позволяет компонентам управлять отображением элементов интерфейса на основе роли и статуса пользователя.

**Листинг 34. Геттеры Vuex-хранилища**

getters: {

isLoading: (state) => state.isLoading, // Текущий статус загрузки

isLoggedIn: (state) => state.user.loggedIn, // Авторизован ли пользователь

userName: (state) => state.user.name, // Имя пользователя

userRole: (state) => state.user.role, // ID роли пользователя

isAdmin: (state) => state.user.role === 1, // Является ли админом

isManager: (state) => state.user.role === 2, // Является ли менеджером

isUser: (state) => state.user.role === 3, // Является ли обычным пользователем

},

Использование Vuex обеспечило централизованное, структурированное и реактивное управление состоянием. За счёт интеграции с localStorage удалось реализовать сохранение пользовательской сессии между обновлениями страницы. Геттеры позволяют эффективно управлять доступом к функциональности в зависимости от роли пользователя, а мутации и действия обеспечивают надёжную обработку авторизации и взаимодействия с сервером.

## Реализация пользовательского интерфейса

**Компонент App**

Файл App отвечает за общую компоновку пользовательского интерфейса интернет-магазина. В нём задаются постоянные элементы, присутствующие на всех страницах приложения, включая шапку, подвал и область отображения содержимого текущего маршрута.

В верхней части расположен компонент MainHeader, который отображает шапку сайта со ссылками навигации, поиском и доступом к профилю и корзине. Нижняя часть содержит компонент MainFooter с контактной информацией и ссылками на основные разделы. Между этими компонентами располагается элемент <router-view />, в котором динамически отображается содержимое активной страницы (например, главная, корзина, страница товара и т.д.).

Для корректного отображения интерфейса на всех устройствах применяется flex-верстка с вертикальной ориентацией и автоматическим заполнением пространства между шапкой и футером. Общий код компонента приведён в листинге 35.

**Листинг 35. Компонент App.vue**

<template>

  <div class="layout">

    <MainHeader />

    <main class="main-content">

      <router-view />

    </main>

    <MainFooter />

  </div>

</template>

<script>

import MainHeader from './components/Headers/MainHeader.vue'

import MainFooter from './components/MainFooter.vue'

export default {

  name: 'App',

  components: {

    MainHeader,

    MainFooter

  }

}

</script>

**Компонент шапки сайта MainHeader**

Компонент MainHeader отвечает за отображение верхней панели интерфейса интернет-магазина «Тайны Востока» и является неотъемлемым элементом всех страниц приложения. Он обеспечивает навигацию, поиск товаров и доступ к модальным окнам, связанным с авторизацией и пользовательским профилем.

Основная структура

Шапка состоит из трёх основных блоков:

* Левый блок содержит название магазина, клик по которому возвращает пользователя на главную страницу.
* Центральный блок включает основные ссылки: «Пункт выдачи», «Корзина», «Профиль», «Заказы», а также кнопку «Инструменты» для администраторов и менеджеров. Последняя визуализирована с помощью иконок шестерёнки, которые меняются при наведении.
* Правый блок реализует строку поиска. Пользователь может ввести запрос и нажать клавишу Enter для запуска поиска. Поисковый запрос обрабатывается через глобальное событие search-request, что позволяет передать его другим компонентам, даже если пользователь находится не на главной странице.

Логика и реактивность

Компонент построен с использованием Composition API и активно применяет:

* Реактивное состояние (ref, computed) для управления отображением модальных окон (showRegister, showLogin, showProfile и др.), а также определения роли пользователя.
* Проверку прав доступа через вычисляемое свойство isAdmin, использующее данные из Vuex-хранилища.
* Логическое управление окнами: например, окно регистрации или профиля открывается в зависимости от текущей авторизации пользователя (store.state.user.loggedIn).
* Интеграцию с Vue Router для обработки переходов, таких как возвращение на главную или переход в корзину.

Компонент управляет отображением следующих модальных окон:

* RegisterModal — регистрация нового пользователя;
* LoginModal — вход в систему;
* ProfileModal — личный кабинет;
* PointsModal — выбор пункта выдачи;
* AdminPanelModal — доступ к административной панели;
* MyOrdersModal — просмотр заказов пользователя.

Каждое модальное окно подключено как отдельный компонент и отображается по соответствующему флагу.

Полный исходный код компонента представлен в листинге 36, а внешний вид шапки показан на рисунке 11.

**Листинг 36. Компонент MainHeader.vue**

<template>

  <header class="header">

    <!-- Основная навигационная панель -->

    <div class="nav-bar">

      <!-- Левый блок: название магазина, переход на главную -->

      <div class="nav-left">

        <a href="#" class="store-name" @click.prevent="goHome">Тайны Востока</a>

      </div>

      <!-- Центр: навигационные ссылки и кнопка администратора -->

      <div class="nav-center">

        <a href="#" class="nav-link" @click.prevent="showDeliveryPoints">Пункт выдачи</a>

        <a href="#" class="nav-link" @click.prevent="goToCart">Корзина</a>

        <a href="#" class="nav-link" @click.prevent="handleProfileClick">Профиль</a>

        <a href="#" class="nav-link" @click.prevent="openOrdersModal">Заказы</a>

        <!-- Кнопка для доступа к панели администратора -->

        <button

          v-if="isAdmin"

          class="nav-link secret-button"

          @click="handleSecretClick"

          title="Инструменты"

        >

          <img src="@/assets/icons/gear-white.svg" alt="Инструменты" class="icon-default" />

          <img src="@/assets/icons/gear-pink.svg" alt="Инструменты" class="icon-hover" />

        </button>

      </div>

      <!-- Правый блок: строка поиска -->

      <div class="nav-right">

        <div class="search-container">

          <input

            type="text"

            class="search-input"

            placeholder="Поиск..."

            v-model="searchQuery"

            @keyup.enter="handleSearch"

          />

        </div>

      </div>

    </div>

    <!-- Модальные окна (по флагам) -->

    <RegisterModal v-if="showRegister" @close="showRegister = false" @switchToLogin="switchToLogin" />

    <LoginModal v-if="showLogin" @close="showLogin = false" @switchToRegister="switchToRegister" @loginSuccess="showProfileModal" />

    <ProfileModal v-if="showProfile" @close="showProfile = false" @logout="handleLogout" />

    <PointsModal v-show="store.state.showDeliveryModal" @close="store.commit('setShowDeliveryModal', false)" @select="handlePointSelect" />

    <AdminPanelModal v-if="showAdminModal" @close="showAdminModal = false" />

    <MyOrdersModal v-if="showOrdersModal" @close="showOrdersModal = false" />

  </header>

</template>

<script>

import RegisterModal from "../auth/RegisterModal.vue";

import LoginModal from "../auth/LoginModal.vue";

import ProfileModal from "../auth/ProfileModal.vue";

import PointsModal from "../Points/PointsModal.vue";

import AdminPanelModal from "../admin/AdminPanelModal.vue";

import MyOrdersModal from "../order/MyOrderModal.vue";

import { useRouter } from "vue-router";

import { ref, computed } from "vue";

import { useStore } from "vuex";

export default {

  name: "MainHeader",

  components: {

    RegisterModal,

    LoginModal,

    ProfileModal,

    PointsModal,

    AdminPanelModal,

    MyOrdersModal,

  },

  setup() {

    const router = useRouter();

    const store = useStore();

    // Модель строки поиска

    const searchQuery = ref("");

    // Флаги для показа модальных окон

    const showRegister = ref(false);

    const showLogin = ref(false);

    const showProfile = ref(false);

    const showAdminModal = ref(false);

    const showOrdersModal = ref(false);

    // Обработка поиска — отправка события

    const handleSearch = () => {

      if (router.currentRoute.value.path !== "/") {

        router.push("/");

      }

      window.dispatchEvent(

        new CustomEvent("search-request", {

          detail: { query: searchQuery.value.trim() },

        })

      );

    };

    // Установка выбранного пункта выдачи

    const handlePointSelect = (point) => {

      store.commit("setSelectedPoint", point);

      store.commit("setShowDeliveryModal", false);

    };

    // Проверка авторизации

    const isAuthenticated = computed(() => store.state.user.loggedIn);

    // Получение роли пользователя

    const userRole = computed(() => store.state.user?.role || null);

    // Проверка, является ли пользователь админом

    const isAdmin = computed(() => {

      const role\_id = Number(userRole.value);

      return role\_id === 1 || role\_id === 2;

    });

    // Открытие модального окна с пунктами выдачи

    const showDeliveryPoints = () => {

      store.commit("setShowDeliveryModal", true);

    };

    return {

      router,

      store,

      searchQuery,

      handleSearch,

      showRegister,

      showLogin,

      showProfile,

      showAdminModal,

      showOrdersModal,

      handlePointSelect,

      showDeliveryPoints,

      isAuthenticated,

      isAdmin,

    };

  },

  methods: {

    // Открыть модальное окно заказов

    openOrdersModal() {

      this.showOrdersModal = true;

    },

    // Переход на главную страницу

    goHome() {

      this.$router.push("/");

    },

    // Открыть модалку администратора

    handleSecretClick() {

      this.showAdminModal = true;

    },

    // Перейти в корзину

    goToCart() {

      this.$router.push("/cart");

    },

    // Показать профиль или окно логина

    handleProfileClick() {

      if (this.store.state.user.loggedIn) {

        this.showProfile = true;

      } else {

        this.showLogin = true;

      }

    },

    // Переключение между логином и регистрацией

    switchToLogin() {

      this.showRegister = false;

      this.showLogin = true;

    },

    switchToRegister() {

      this.showLogin = false;

      this.showRegister = true;

    },

    // Показать профиль после логина

    showProfileModal() {

      this.showLogin = false;

      this.showProfile = true;

    },

    // Выход из профиля

    handleLogout() {

      this.store.dispatch("logout");

      this.showProfile = false;

    },

  },

};

</script>

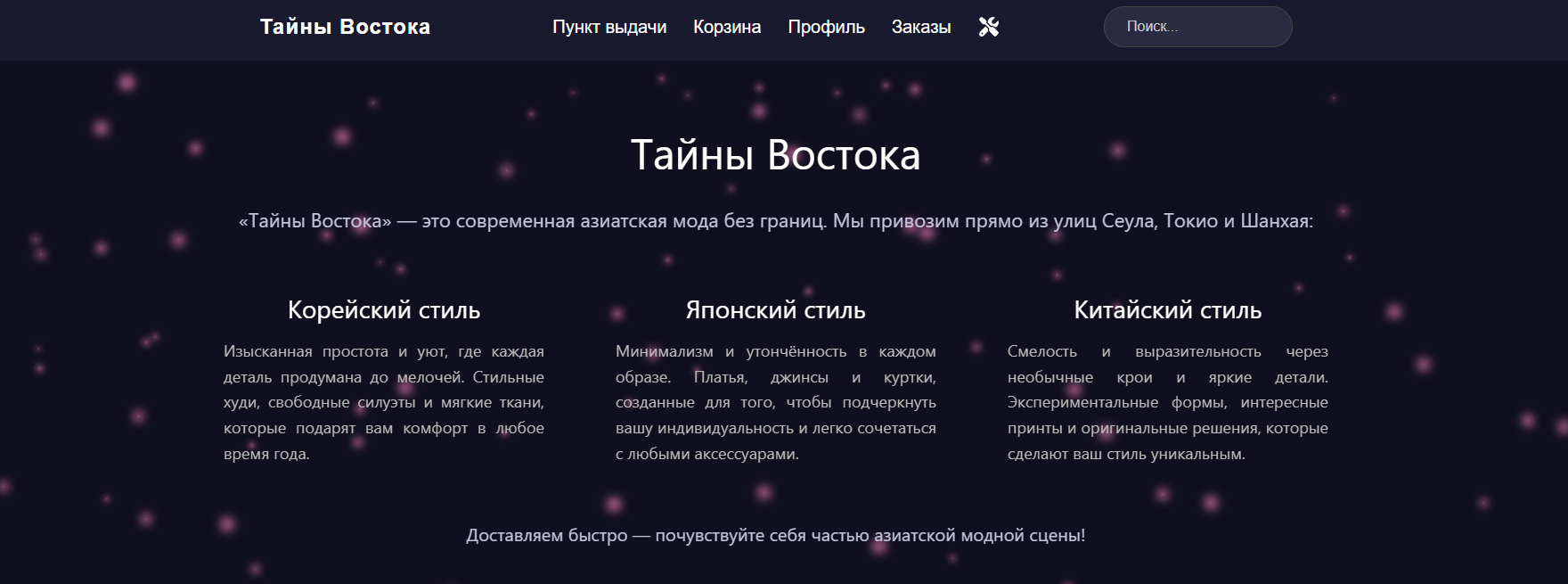


Рисунок 11 – Шапка сайта MainHeader и HeaderInfo

**Компонент модального окна входа (LoginModal)**

Компонент LoginModal предназначен для авторизации пользователей в приложении. Он представляет собой модальное окно с формой, содержащей два обязательных поля: для ввода email и пароля. При вводе некорректных данных или ошибках сервера пользователю отображается соответствующее сообщение об ошибке. Авторизация происходит асинхронно через Vuex-экшен, который отправляет запрос к API.

Пользователь может закрыть окно по кнопке «×» или кликнув на затемнённый фон, что обеспечивает удобство и привычное поведение модальных окон. Кроме того, внизу формы есть ссылка для перехода к регистрации, которая переключает текущее окно на модальное окно регистрации.

Для обработки ошибок реализована логика, которая различает ошибки валидации, ошибки авторизации и серверные сбои, предоставляя пользователю понятные и информативные сообщения.

Подробный код компонента приведён в листинге 37, а визуальное оформление модального окна показано на рисунке 12.

**Листинг 37. Компонент LoginModal.vue**

<template>

  <!-- Оверлей модального окна авторизации -->

  <div class="modal-overlay" @click.self="close">

    <div class="modal">

      <!-- Кнопка закрытия -->

      <button class="close-btn" @click="close">×</button>

      <h2 class="modal-title">Авторизация</h2>

      <form @submit.prevent="login">

        <!-- Ввод email -->

        <input v-model="email" type="email" placeholder="Email" required />

        <!-- Ввод пароля -->

        <input v-model="password" type="password" placeholder="Пароль" required />

        <!-- Сообщение об ошибке -->

        <div v-if="errorMessage" class="error-message">*{{ errorMessage }}*</div>

        <!-- Кнопка входа -->

        <button type="submit" class="submit-btn">Войти</button>

      </form>

      <!-- Ссылка для перехода к регистрации -->

      <p class="switch-to-register">

        Нет аккаунта?

        <span @click="switchToRegister" class="link">Зарегистрироваться</span>

      </p>

    </div>

  </div>

</template>

<script>

export default {

  props: ["show"],

  emits: ["close", "switchToRegister"],

  data() {

    return {

      email: "",

      password: "",

      errorMessage: "", // Сюда будет записываться сообщение об ошибке

    };

  },

  methods: {

    // Метод авторизации

    async login() {

      this.errorMessage = ""; // Сброс ошибки перед отправкой

      try {

        // Отправка запроса на login через Vuex

        await this.$store.dispatch("login", {

          email: this.email,

          password: this.password,

        });

        // Закрываем модалку после успешного входа

        this.$emit("close");

      } catch (error) {

        // Обработка ошибок от сервера (Laravel)

        if (error.response) {

          const { status, data } = error.response;

          // Ошибка через ApiException (например, 401)

          if (data.message) {

            this.errorMessage = data.message;

          }

          // Ошибка валидации формы (422)

          else if (status === 422 && data.errors) {

            const firstField = Object.keys(data.errors)[0];

            this.errorMessage = data.errors[firstField][0];

          } else {

            this.errorMessage = "Ошибка сервера. Попробуйте позже.";

          }

        } else {

          this.errorMessage = "Сервер не отвечает.";

        }

      }

    },

    // Переход на регистрацию

    switchToRegister() {

      this.$emit("switchToRegister");

      this.$emit("close");

    },

    // Закрытие окна

    close() {

      this.$emit("close");

    },

  },

};

</script>

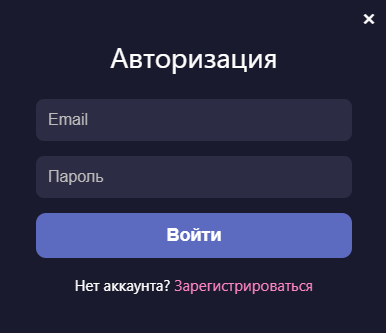
****

Рисунок 12 – Модальное окно авторизации LoginModal

**Компонент модального окна регистрации (RegisterModal)**

Компонент RegisterModal реализует модальное окно для регистрации новых пользователей. В его составе — форма с обязательными полями для ввода ФИО, email, телефона, пароля и выбора пола. Пароль можно временно показать или скрыть с помощью отдельной кнопки, что повышает удобство ввода.

При отправке данных форма асинхронно взаимодействует с API, обрабатывая возможные ошибки валидации и отображая их рядом с соответствующими полями. Если сервер возвращает общие ошибки, они показываются через alert. Окно можно закрыть кнопкой «×» или кликом по затемнённому фону. Для удобства пользователя предусмотрена ссылка, позволяющая переключиться на форму входа.

После успешной регистрации в локальное хранилище сохраняется токен авторизации, и автоматически открывается профиль пользователя. Это обеспечивает плавный и понятный процесс регистрации и входа.

Подробный код компонента представлен в листинге 38, а визуальное оформление показано на рисунке 13.

**Листинг 38. Компонент RegisterModal.vue**

<template>

  <!-- Оверлей, закрывается при клике вне модального окна -->

  <div class="modal-overlay" @click.self="close">

    <div class="modal">

      <!-- Кнопка закрытия окна -->

      <button class="close-btn" @click="close">×</button>

      <h2 class="modal-title">Регистрация</h2>

      <!-- Форма регистрации -->

      <form @submit.prevent="register">

        <input v-model="name" type="text" placeholder="ФИО" required />

        <div v-if="errors.name" class="error-message">{{ errors.name[0] }}</div>

        <input v-model="email" type="email" placeholder="Email" required />

        <div v-if="errors.email" class="error-message">{{ errors.email[0] }}</div>

        <input v-model="phone" type="text" placeholder="Телефон" required />

        <div v-if="errors.telephone" class="error-message">{{ errors.telephone[0] }}</div>

        <!-- Блок с паролем и кнопкой показа/скрытия -->

        <div class="password-wrapper">

          <input

            :type="showPassword ? 'text' : 'password'"

            v-model="password"

            placeholder="Пароль"

            required

          />

          <button type="button" class="toggle-password" @click="togglePassword">

            {{ showPassword ? 'Скрыть' : 'Показать' }}

          </button>

        </div>

        <div v-if="errors.password" class="error-message">{{ errors.password[0] }}</div>

        <select v-model="gender" required>

          <option value="male">Мужской</option>

          <option value="female">Женский</option>

        </select>

        <div v-if="errors.gender" class="error-message">{{ errors.gender[0] }}</div>

        <button type="submit" class="submit-btn">Зарегистрироваться</button>

      </form>

      <!-- Переключение на форму авторизации -->

      <p class="switch-to-login">

        Уже есть аккаунт? <span @click="switchToLogin" class="link">Авторизация</span>

      </p>

    </div>

  </div>

</template>

<script>

import api from "@/services/api";

export default {

  props: ["show"],

  emits: ["close", "switchToLogin", "openProfile"],

  data() {

    return {

      name: "",

      email: "",

      phone: "",

      password: "",

      gender: "male",

      errors: {}, // Хранит ошибки валидации от сервера

      showPassword: **false**, // Переключатель видимости пароля

    };

  },

  methods: {

    // Переключение видимости пароля

    togglePassword() {

      this.showPassword = !this.showPassword;

    },

    // Отправка данных на сервер для регистрации

    async register() {

      this.errors = {}; // Сброс ошибок перед запросом

      try {

        // Подготовка объекта с данными для API

        const requestData = {

          name: this.name,

          email: this.email,

          telephone: this.phone || **null**, // Телефон может отсутствовать

          sex: this.gender === "male", // Преобразуем в boolean для API

          password: this.password,

        };

        const response = await api.post("/register", requestData);

        if (response.status === 201) {

          // Сохраняем токен и закрываем модалку

          localStorage.setItem("auth\_token", response.data.token);

          alert("Вы успешно зарегистрированы!");

          this.$emit("close");

          this.$emit("openProfile"); // Открываем профиль после регистрации

        }

      } catch (error) {

        if (error.response) {

          // Валидационные ошибки от API

          const serverErrors = error.response.data.errors;

          if (serverErrors) {

            this.errors = serverErrors;

          } else {

            alert(error.response.data.message || "Ошибка регистрации");

          }

        } else {

          alert("Что-то пошло не так");

        }

      }

    },

    // Переход на форму авторизации

    switchToLogin() {

      this.$emit("switchToLogin");

      this.$emit("close");

    },

    // Закрытие модального окна

    close() {

      this.$emit("close");

    },

  },

};

</script>

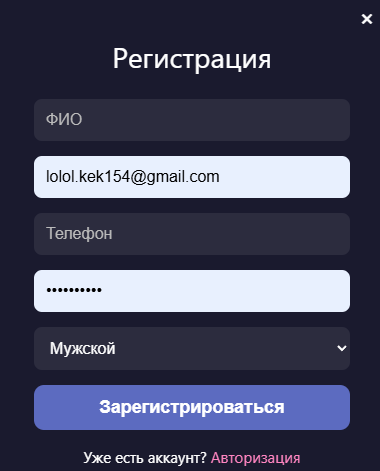
****

Рисунок 13 - Модальное окно регистрации RegisterModal

**Компонент модального окна профиля (ProfileModal)**

Данный компонент отображает информацию о текущем пользователе и предоставляет возможность редактирования его данных. В модальном окне выводятся поля для ФИО, email и телефона, а также кнопки «Редактировать» и «Выйти». При нажатии на «Редактировать» поля становятся доступны для изменения, и появляется возможность сохранить или отменить внесённые правки.

Все изменения отправляются асинхронно на сервер через API с обработкой возможных ошибок валидации, которые выводятся рядом с соответствующими полями. Если при загрузке данных или сохранении произошли критические ошибки, отображается сообщение и даётся возможность повторить попытку. Выход из системы происходит с очисткой токена и уведомлением родительского компонента.

Модальное окно закрывается нажатием на кнопку «×» или кликом на затемнённый фон. При наличии не сохранённых изменений пользователь получает предупреждение перед закрытием.

Подробный код компонента приведён в листинге 39, визуальное оформление — на рисунке 14.

**Листинг 39. Компонент ProfileModal.vue**

<template>

  <div v-if="isVisible" class="modal-overlay" @click.self="handleClose">

    <div class="modal-content">

      <h2>Профиль</h2>

      <button @click="handleClose" class="close-btn">×</button>

      <div v-if="loading" class="loading-state">

        <div class="spinner"></div>

        <p>Загрузка данных...</p>

      </div>

      <div v-else-if="user && !isEditing" class="profile-info">

        <div class="avatar-placeholder">*{{ getInitials(user.name) }}*</div>

        <div class="info-item"><span class="label">Имя:</span> <span>*{{ user.name }}*</span></div>

        <div class="info-item"><span class="label">Email:</span> <span>*{{ user.email }}*</span></div>

        <div v-if="user.telephone" class="info-item"><span class="label">Телефон:</span> <span>*{{ user.telephone }}*</span></div>

        <div class="actions">

          <button @click="startEditing" class="edit-btn">Редактировать</button>

          <button @click="logout" class="logout-btn">Выйти</button>

        </div>

      </div>

      <div v-else-if="isEditing" class="edit-form">

        <div class="form-group">

          <label for="name">Имя:</label>

          <input

            id="name"

            v-model="editForm.name"

            type="text"

            class="form-input"

          />

          <!-- Ошибка с применением стиля из примера -->

          <div v-if="validationErrors.name" class="error-message">{{ validationErrors.name[0] }}</div>

        </div>

        <div class="form-group">

          <label for="email">Email:</label>

          <input

            id="email"

            v-model="editForm.email"

            type="email"

            class="form-input"

          />

          <div v-if="validationErrors.email" class="error-message">{{ validationErrors.email[0] }}</div>

        </div>

        <div class="form-group">

          <label for="telephone">Телефон:</label>

          <input

            id="telephone"

            v-model="editForm.telephone"

            type="tel"

            class="form-input"

          />

          <div v-if="validationErrors.telephone" class="error-message">{{ validationErrors.telephone[0] }}</div>

        </div>

        <div class="form-actions">

          <button @click="saveChanges" class="save-btn" :disabled="isSaving">

            {{ isSaving ? "Сохранение..." : "Сохранить" }}

          </button>

          <button @click="cancelEditing" class="cancel-btn">Отмена</button>

        </div>

        <!-- Общая ошибка -->

        <div v-if="errorMessage" class="error-message" style="margin-top: 15px;">

*{{ errorMessage }}*

        </div>

      </div>

      <div v-else class="error-state">

        <p class="error-message">⚠️ Не удалось загрузить данные профиля</p>

        <button @click="retryLoading" class="retry-btn">Повторить попытку</button>

        <button @click="logout" class="logout-btn">Выйти</button>

      </div>

    </div>

  </div>

</template>

<script>

import api from "@/services/api";

export default {

  name: "ProfileModal",

  data() {

    return {

      user: **null**,

      isVisible: **true**,

      loading: **true**,

      error: **null**,

      isEditing: **false**,

      isSaving: **false**,

      editForm: {

        name: "",

        email: "",

        telephone: ""

      },

      validationErrors: {} // Ошибки валидации для полей формы

    };

  },

  async mounted() {

    await this.loadUserData();

  },

  methods: {

    // Загрузка данных пользователя

    async loadUserData() {

      try {

        this.loading = true;

        this.error = null;

        const token = localStorage.getItem("auth\_token");

        if (!token) throw new Error("Токен отсутствует");

        const response = await api.get("/profile", {

          headers: {

            Authorization: `Bearer ${token}`

          }

        });

        this.user = response.data;

        this.editForm = {

          name: this.user.name,

          email: this.user.email,

          telephone: this.user.telephone || ""

        };

      } catch (error) {

        console.error("Ошибка при загрузке профиля:", error);

        this.error = error.response?.data?.message || error.message;

        if (error.response?.status === 401) this.handleForceLogout();

      } finally {

        this.loading = false;

      }

    },

    // Получение инициалов пользователя для аватара

    getInitials(name) {

      return name

        .split(" ")

        .map((part) => part[0])

        .join("")

        .toUpperCase();

    },

    // Начать редактирование

    startEditing() {

      this.isEditing = true;

      this.validationErrors = {}; // очистка ошибок при начале редактирования

    },

    // Отмена редактирования, восстановление данных

    cancelEditing() {

      this.isEditing = false;

      this.validationErrors = {};

      this.editForm = {

        name: this.user.name,

        email: this.user.email,

        telephone: this.user.telephone || ""

      };

    },

    // Сохранение изменений с обработкой ошибок валидации

    async saveChanges() {

      try {

        this.isSaving = true;

        this.validationErrors = {}; // очищаем ошибки перед отправкой

        const token = localStorage.getItem("auth\_token");

        const response = await api.patch("/profile", this.editForm, {

          headers: {

            Authorization: `Bearer ${token}`

          }

        });

        const updatedUser = response.data.data;

        // Обновляем данные пользователя в компоненте

        this.user.name = updatedUser.name;

        this.user.email = updatedUser.email;

        this.user.telephone = updatedUser.telephone;

        this.isEditing = false;

        this.$emit("profile-updated", this.user);

      } catch (error) {

        this.validationErrors = {};

        if (error.response && error.response.data) {

          const data = error.response.data;

          if (data.errors) {

            // Записываем ошибки валидации в объект для показа в форме

            this.validationErrors = data.errors;

          } else if (data.message) {

            alert(data.message);

          } else {

            alert("Ошибка при сохранении данных");

          }

        } else {

          alert("Ошибка при сохранении данных");

        }

      } finally {

        this.isSaving = false;

      }

    },

    // Выход из профиля

    logout() {

      localStorage.removeItem("auth\_token");

      this.$emit("logout");

      this.$emit("close");

    },

    // Принудительный выход при 401

    handleForceLogout() {

      this.logout();

    },

    // Повторная загрузка данных

    retryLoading() {

      this.loadUserData();

    },

    // Закрытие модального окна с подтверждением если есть несохранённые изменения

    handleClose() {

      if (

        this.isEditing &&

        !confirm("У вас есть несохраненные изменения. Закрыть без сохранения?")

      ) {

        return;

      }

      this.$emit("close");

    }

  }

};

</script>

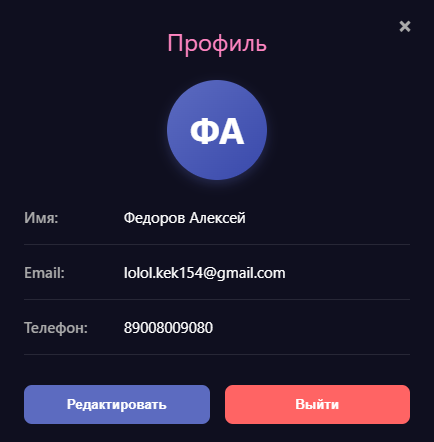
****

Рисунок 14 - Модальное окно профиля ProfileModal

**Компонент подвала сайта (MainFooter)**

Этот компонент отвечает за отображение подвала сайта с двумя кнопками: «О нас» и «Контакты». При нажатии на каждую из кнопок открывается соответствующее модальное окно с информацией. Окно «О нас» содержит краткое описание магазина, его концепцию и направление. Окно «Контакты» показывает данные разработчиков , включая телефоны и ссылки на их профили в социальных сетях ВКонтакте, Telegram и GitHub.

Модальные окна можно закрыть кликом по затемнённому фону или кнопкой «×». Такое решение обеспечивает удобный и понятный интерфейс для посетителей сайта.

Данный компонент визуально представлен на Рисунках 15, 16 и 17.

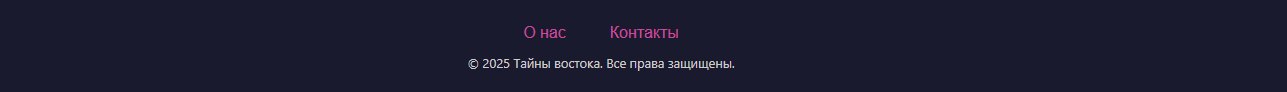
****

Рисунок 15 – Компонент подвала MainFooter

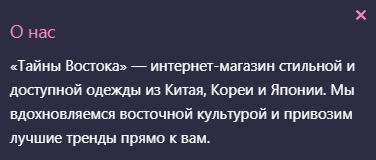
****

Рисунок 16 – Окно о нас MainFooter

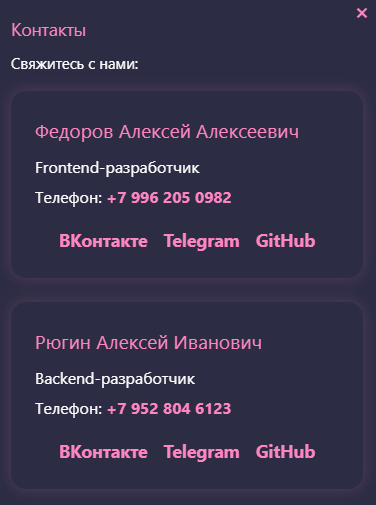
****

Рисунок 17 – Окно контактов MainFooter

**Представление** **HomeView**

В представлении HomeView реализован базовый макет интернет-магазина, состоящий из трёх ключевых частей.

Сверху размещён компонент HeaderInfo, который служит информационным блоком с основными данными или акцентами сайта.

Основной контент разделён на две части по горизонтали:

* Левая часть — это компонент ProductFilter, отвечающий за фильтрацию товаров по различным параметрам (категории, цена, цвет и т.п.). Пользователь может выбрать нужные фильтры, и они передаются в компонент отображения товаров.
* Правая часть — блок ProductCards, в котором отображаются карточки товаров с учётом текущих фильтров и поискового запроса. Компонент получает объект фильтров и строку поиска и обновляет список товаров динамически.

Связь между фильтрами и карточками товаров реализована через передачу данных с помощью реактивных переменных. Обновление поискового запроса происходит по событию из шапки сайта, что обеспечивает единое управление отображаемыми товарами.

Такой подход позволяет удобно разделить логику фильтрации и отображения, а также облегчает масштабирование и поддержку кода. Полный код компонента приведён в листинге 40.

**Листинг 40. Представление HomeView.vue**

<template>

  <HeaderInfo></HeaderInfo>

  <div class="home-container">

    <!-- Левая часть — фильтрация -->

    <ProductFilter @filter-changed="handleFilterChange" />

    <!-- Правая часть — карточки товаров -->

    <div class="products-section">

      <ProductCards

        :filters="activeFilters"

        :search-query="searchQuery"

      />

    </div>

  </div>

</template>

<script setup>

import {onMounted, ref} from 'vue'

import ProductFilter from '@/components/Product/ProductFilter.vue'

import ProductCards from '@/components/Product/ProductCards.vue'

import HeaderInfo from "@/components/Headers/HeaderInfo.vue";

const activeFilters = ref({});

const searchQuery = ref('');

const handleFilterChange = (filters) => {

  activeFilters.value = filters;

};

onMounted(() => {

  window.addEventListener('search-request', (event) => {

    searchQuery.value = event.detail.query;

  });

});

</script>

**Компонент** **ProductCards**

Компонент ProductCards отвечает за отображение списка товаров на странице каталога интернет-магазина. Он обеспечивает удобный и понятный интерфейс для пользователя, позволяя просматривать товары с учётом фильтров и поискового запроса, а также переключаться между страницами с помощью пагинации.

При загрузке данных с сервера компонент показывает индикатор загрузки, чтобы пользователь видел, что происходит запрос. Если по выбранным параметрам товары не найдены, выводится сообщение с предложением изменить фильтры. В случае успешной загрузки отображается сетка карточек товаров.

Визуальное представление компонента показано на рисунке 18, а полный исходный код компонента приведён в листинге 41.

Основные особенности компонента:

1. Загрузка и обновление товаров

* Данные загружаются с сервера с учётом активных фильтров и поискового запроса.
* При изменении фильтров или поисковой строки выполняется автоматический повторный запрос.
* В процессе загрузки отображается индикатор загрузки, а при отсутствии результатов — понятное сообщение.

1. Отображение карточек товаров

* Каждая карточка содержит изображение товара, название, страну производителя, категорию и цену.
* Изображения предварительно загружаются с обработкой состояний (загрузка, успешная загрузка, ошибка).
* Для товаров без наличия отображается оверлей с сообщением «Нет в наличии» и отключается переход по клику.
* По клику на карточку товара, если товар в наличии, происходит переход на страницу с подробной информацией.

1. Пагинация

* Отображается максимум 12 товаров на странице.
* Пагинация реализована с кнопками для перехода между страницами.
* Для удобства навигации показывается ограниченное количество номеров страниц.
* При смене страницы происходит плавная прокрутка к началу списка товаров.

1. Редактирование товаров для администраторов и менеджеров

* Для пользователей с ролью администратора или менеджера на карточках отображается кнопка «Редактировать».
* При нажатии на эту кнопку открывается модальное окно редактирования товара.
* После сохранения изменений список товаров автоматически обновляется.

**Листинг 41. Компонент ProductCards.vue**

<template>

  <div class="products-container" ref="productsContainer">

    <!-- Состояние загрузки -->

    <div v-if="loading" class="loading-overlay">

      <div class="loader"></div>

    </div>

    <!-- Сообщение если нет товаров -->

    <div v-if="!loading && products.length === 0" class="no-products">

      <h3>Товары не найдены</h3>

      <p>Попробуйте изменить параметры фильтрации</p>

    </div>

    <!-- Сетка товаров -->

    <div v-else class="products-grid">

      <div

        v-for="product in paginatedProducts"

        :key="product.id"

        class="product-card"

        :class="{ 'out-of-stock': getProductTotalQuantity(product) === 0 }"

      >

        <div class="product-image-container" @click="handleProductClick(product)">

          <!-- Состояние загрузки изображения -->

          <div v-if="imageStates[product.id] === 'loading'" class="image-loading-skeleton"></div>

          <!-- Основное изображение товара -->

          <img

            v-if="imageStates[product.id] === 'loaded'"

            :src="getImageUrl(product)"

            :alt="product.name"

            class="product-image"

          />

          <!-- Заглушка при ошибке загрузки изображения -->

          <div v-if="imageStates[product.id] === 'error'" class="image-placeholder-container">

            <img

              :src="placeholderImage"

              :alt="product.name"

              class="placeholder-image"

            />

            <div class="placeholder-overlay">

              <span class="placeholder-text">Фото недоступно</span>

            </div>

          </div>

          <!-- Оверлей для товаров, которых нет в наличии -->

          <div v-if="getProductTotalQuantity(product) === 0" class="out-of-stock-overlay">

            <span class="out-of-stock-text">Нет в наличии</span>

          </div>

        </div>

        <!-- Информация о товаре -->

        <div class="product-info" @click="handleProductClick(product)">

          <div class="product-meta">

            <span class="product-country">*{{ product.country?.name }}*</span>

            <span class="product-category">*{{ product.category?.name }}*</span>

          </div>

          <h3 class="product-name">*{{ product.name }}*</h3>

          <div class="product-price">

            <span class="current-price">*{{ formatPrice(product.price) }}*</span>

          </div>

        </div>

        <!-- Кнопка редактирования (видна только админам/менеджерам) -->

        <button

          v-if="isAdminOrManager"

          class="edit-button"

          @click.stop="openEditModal(product)"

          type="button"

        >

          Редактировать

        </button>

      </div>

    </div>

    <!-- Пагинация -->

    <div v-if="totalPages > 1 && !loading" class="pagination">

      <button

        @click="prevPage"

        :disabled="currentPage === 1"

        class="pagination-btn"

      >

        &lt;

      </button>

      <button

        v-for="page in visiblePages"

        :key="page"

        @click="goToPage(page)"

        :class="{ active: currentPage === page }"

        class="pagination-btn"

      >

*{{ page }}*

      </button>

      <button

        @click="nextPage"

        :disabled="currentPage === totalPages"

        class="pagination-btn"

      >

        &gt;

      </button>

    </div>

    <!-- Модальное окно редактирования товара -->

    <EditProductModal

      v-if="showEditModal"

      :productId="editingProductId"

      @close="closeEditModal"

      @updated="onProductUpdated"

    />

  </div>

</template>

<script setup>

import { ref, computed, onMounted, onUnmounted, watch } from 'vue';

import { useRouter } from 'vue-router';

import api from '@/services/api';

import { useStore } from 'vuex';

import EditProductModal from '../admin/EditProductModal.vue';

import placeholderImage from '@/assets/images/product-placeholder.jpeg';

// Пропсы компонента

const props = defineProps({

  filters: {

    type: Object,

    default: () => ({})

  }

});

// Реактивные переменные

const products = ref([]);

const loading = ref(false);

const currentPage = ref(1);

const itemsPerPage = 12; // Количество товаров на странице

const showEditModal = ref(false);

const editingProductId = ref(null);

const imageStates = ref({}); // Состояния загрузки изображений

const searchTerm = ref(''); // Поисковый запрос

const router = useRouter();

const store = useStore();

// Получаем роль пользователя из хранилища

const userRole = computed(() => store.state.user?.role || null);

// Проверяем, является ли пользователь администратором или менеджером

const isAdminOrManager = computed(() => {

  const role\_id = Number(userRole.value);

  return role\_id === 1 || role\_id === 2;

});

// Формируем URL изображения товара

const getImageUrl = (product) => {

  if (!product.photo) return '';

  return product.photo.startsWith('http')

    ? product.photo

    : `http://secrets-of-the-east.ru/storage/${product.photo}`;

};

// Предзагрузка изображений товаров

const preloadImages = (products) => {

  products.forEach(product => {

    const img = new Image();

    const imageUrl = getImageUrl(product);

    if (!imageUrl) {

      imageStates.value[product.id] = 'error';

      return;

    }

    imageStates.value[product.id] = 'loading';

    img.src = imageUrl;

    img.onload = () => {

      imageStates.value[product.id] = 'loaded';

    };

    img.onerror = () => {

      imageStates.value[product.id] = 'error';

    };

  });

};

// Получаем общее количество товара (учитывая варианты)

const getProductTotalQuantity = (product) => {

  if (product.quantity !== undefined) {

    return product.quantity;

  }

  if (product.product\_color\_sizes && product.product\_color\_sizes.length > 0) {

    return product.product\_color\_sizes.reduce((sum, item) => sum + item.quantity, 0);

  }

  return 0;

};

// Обработчик клика по товару

const handleProductClick = (product) => {

  if (getProductTotalQuantity(product) === 0) return;

  goToProduct(product.id);

};

// Загрузка товаров с сервера

const fetchProducts = async () => {

  loading.value = true;

  imageStates.value = {};

  try {

    let searchResults = [];

    let filterResults = [];

    // Если есть поисковый запрос - выполняем поиск

    if (searchTerm.value) {

      const searchRes = await api.get('/products/search', {

        params: { q: searchTerm.value }

      });

      searchResults = searchRes.data.data;

    }

    // Загружаем товары по фильтрам

    const filterRes = await api.get('/products/filter', {

      params: cleanParams(props.filters)

    });

    filterResults = filterRes.data.data;

    // Объединяем результаты поиска и фильтрации

    let combinedProducts = [];

    if (searchTerm.value) {

      const searchIds = searchResults.map(p => p.id);

      combinedProducts = filterResults.filter(p => searchIds.includes(p.id));

    } else {

      combinedProducts = filterResults;

    }

    // Сортируем товары: сначала те, что в наличии

    combinedProducts.sort((a, b) => {

      const aQty = getProductTotalQuantity(a);

      const bQty = getProductTotalQuantity(b);

      if (aQty === 0 && bQty > 0) return 1;

      if (aQty > 0 && bQty === 0) return -1;

      return 0;

    });

    products.value = combinedProducts;

    preloadImages(combinedProducts);

  } catch (e) {

    console.error('Ошибка загрузки товаров:', e);

    products.value = [];

  } finally {

    loading.value = false;

  }

};

// Вычисляемые свойства для пагинации

const totalPages = computed(() => Math.ceil(products.value.length / itemsPerPage));

const paginatedProducts = computed(() =>

  products.value.slice((currentPage.value - 1) \* itemsPerPage, currentPage.value \* itemsPerPage)

);

const visiblePages = computed(() => {

  const pages = [];

  const maxVisible = 5;

  let start = Math.max(1, currentPage.value - Math.floor(maxVisible / 2));

  let end = Math.min(totalPages.value, start + maxVisible - 1);

  if (end - start + 1 < maxVisible) {

    start = Math.max(1, end - maxVisible + 1);

  }

  for (let i = start; i <= end; i++) {

    pages.push(i);

  }

  return pages;

});

// Навигация по страницам

const productsContainer = ref(null);

const scrollToComponent = () => {

  if (productsContainer.value) {

    // Вариант 1: Просто к верху компонента

    productsContainer.value.scrollIntoView({

      behavior: 'smooth',

      block: 'start'

    });

    // ИЛИ Вариант 2: С учетом фиксированного хедера (если есть)

    const headerHeight = 80; // Высота вашего хедера

    const topPos = productsContainer.value.offsetTop - headerHeight;

    window.scrollTo({

      top: topPos,

      behavior: 'smooth'

    });

  }

};

const goToPage = (page) => {

  if (page !== currentPage.value) {

    currentPage.value = page;

    scrollToComponent();

  }

};

const prevPage = () => {

  if (currentPage.value > 1) {

    currentPage.value--;

    scrollToComponent();

  }

};

const nextPage = () => {

  if (currentPage.value < totalPages.value) {

    currentPage.value++;

    scrollToComponent();

  }

};

// Форматирование цены

const formatPrice = (price) => {

  return new Intl.NumberFormat('ru-RU', {

    style: 'currency',

    currency: 'RUB',

    maximumFractionDigits: 0

  }).format(price);

};

// Переход на страницу товара

const goToProduct = (id) => {

  router.push(`/product/${id}`);

};

// Управление модальным окном редактирования

const openEditModal = (product) => {

  editingProductId.value = product.id;

  showEditModal.value = true;

};

const closeEditModal = () => {

  showEditModal.value = false;

  editingProductId.value = null;

};

const onProductUpdated = () => {

  closeEditModal();

  fetchProducts();

};

// Очистка параметров от пустых значений

const cleanParams = (obj) => {

  const res = {};

  for (const key in obj) {

    const val = obj[key];

    if (val !== **null** && val !== undefined && val !== '') {

      res[key] = val;

    }

  }

  return res;

};

// Обработчик поискового запроса

const handleSearch = (event) => {

  searchTerm.value = event.detail.query || '';

};

// Наблюдатели за изменениями

watch(() => props.filters, () => {

  currentPage.value = 1;

  fetchProducts();

}, { deep: **true** });

watch(searchTerm, () => {

  currentPage.value = 1;

  fetchProducts();

});

onMounted(() => {

  window.addEventListener('search-request', handleSearch);

  fetchProducts();

});

onUnmounted(() => {

  window.removeEventListener('search-request', handleSearch);

});

</script>

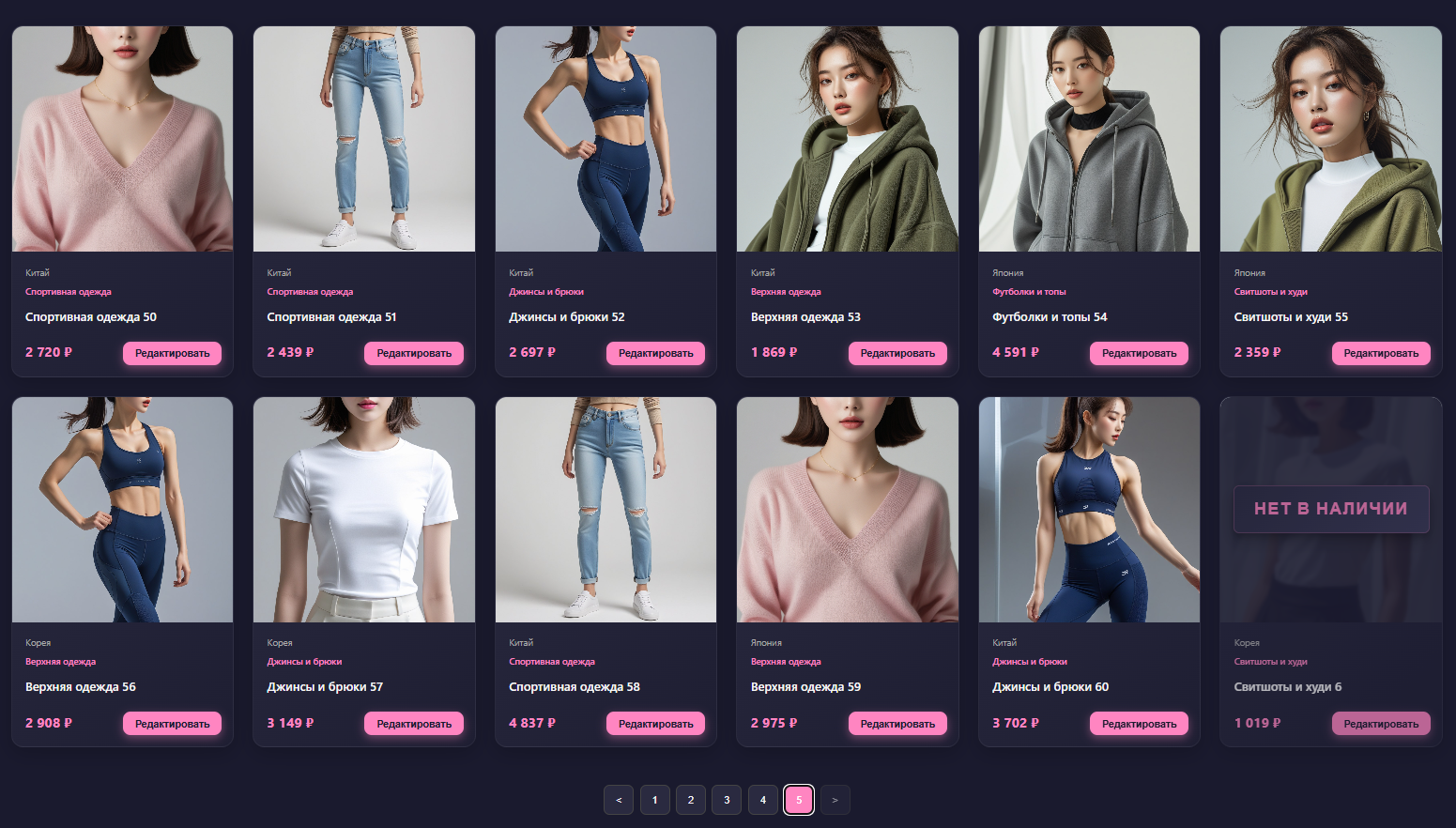


Рисунок 18 -Компонент карточек товара ProductCards

**Компонент ProductFilter**

Компонент ProductFilter предоставляет удобный интерфейс для фильтрации товаров в каталоге интернет-магазина. Он позволяет пользователю выбрать категорию, страну производства, пол и диапазон цены для отбора товаров. Компонент загружает данные для фильтров с сервера при инициализации и отправляет выбранные параметры в родительский компонент для обновления списка товаров.

Визуальное представление компонента показано на рисунке 19, а полный исходный код приведён в листинге 42.

Основные особенности компонента:

1. Загрузка данных для фильтров

* При создании компонента происходит асинхронный запрос категорий и стран через API.
* В случае ошибки загрузки отображается информативное сообщение.

1. Выбор фильтров

* Пользователь может выбрать одну категорию, одну страну и пол (мужской, женский или любой).
* Для цены доступен диапазон с вводом минимального и максимального значения.
* Компонент контролирует корректность диапазона, не позволяя минимальной цене превышать максимальную.

1. Применение фильтров

* По нажатию кнопки «Применить фильтры» формируется объект с выбранными параметрами.
* Пустые или стандартные значения не включаются в объект.
* Выбранные фильтры отправляются через событие filter-changed родительскому компоненту.

1. Сброс фильтров

* Кнопка «Сбросить фильтры» возвращает значения фильтров к начальным (пустым).
* Отправляет пустой объект через событие filter-changed, что приводит к показу всех товаров.

**Листинг 42. Компонент ProductFilter.vue**

<template>

  <div class="filter-container">

    <h2>Фильтры</h2>

    <!-- Ошибка загрузки фильтров -->

    <div v-if="errors.load" class="error-message">*{{ errors.load }}*</div>

    <!-- Фильтр по категориям -->

    <div class="filter-section" :class="sectionClass">

      <h3>Категории</h3>

      <select v-model="selectedCategory" class="filter-select">

        <option value="">Все категории</option>

        <option v-for="category in categories" :key="category.id" :value="category.id">

*{{ category.name }}*

        </option>

      </select>

    </div>

    <!-- Фильтр по странам -->

    <div class="filter-section" :class="sectionClass">

      <h3>Страны</h3>

      <select v-model="selectedCountry" class="filter-select">

        <option value="">Все страны</option>

        <option v-for="country in countries" :key="country.id" :value="country.id">

*{{ country.name }}*

        </option>

      </select>

    </div>

    <!-- Фильтр по полу -->

    <div class="filter-section" :class="sectionClass">

      <h3>Пол</h3>

      <select v-model="selectedGender" class="filter-select">

        <option value="">Не важно</option>

        <option value="0">Мужской</option>

        <option value="1">Женский</option>

      </select>

    </div>

    <!-- Фильтр по цене -->

    <div class="filter-section" :class="sectionClass">

      <h3>Цена</h3>

      <div class="price-range">

        <div class="price-inputs">

          <input

            type="number"

            v-model.number="priceRange[0]"

            :min="minPrice"

            :max="maxPrice"

            @input="updatePriceText"

            class="price-input"

            placeholder="От"

          />

          <span>—</span>

          <input

            type="number"

            v-model.number="priceRange[1]"

            :min="minPrice"

            :max="maxPrice"

            @input="updatePriceText"

            class="price-input"

            placeholder="До"

          />

        </div>

        <div class="price-values">

          <span>{{ priceRange[0] }} ₽</span>

          <span>{{ priceRange[1] }} ₽</span>

        </div>

      </div>

    </div>

    <!-- Кнопки применения и сброса -->

    <button @click="applyFilters" class="apply-btn">Применить фильтры</button>

    <button @click="resetFilters" class="reset-btn">Сбросить фильтры</button>

  </div>

</template>

<script setup>

import { ref, onMounted } from 'vue'

import api from "@/services/api"

// Хранилища для выбранных фильтров

const selectedCategory = ref('')

const selectedCountry = ref('')

const selectedGender = ref('')

const priceRange = ref([0, 100000])

// Границы диапазона цены

const minPrice = 0

const maxPrice = 100000

// Данные для селектов

const categories = ref([])

const countries = ref([])

// Ошибки загрузки данных фильтра

const errors = ref({})

// Эмит события с фильтрами

const emit = defineEmits(['filter-changed'])

// Загрузка категорий и стран с API

const loadFilterData = async () => {

  errors.value.load = **null**

  try {

    const [cats, cntrs] = await Promise.all([

      api.get('/category'),

      api.get('/country')

    ])

    categories.value = cats.data

    countries.value = cntrs.data

  } catch (error) {

    errors.value.load = 'Не удалось загрузить данные фильтров. Попробуйте обновить страницу.'

  }

}

// Отправляем выбранные фильтры наружу

const applyFilters = () => {

  const filters = {

    category\_id: selectedCategory.value || **null**,

    country\_id: selectedCountry.value || **null**,

    min\_price: priceRange.value[0] !== minPrice ? priceRange.value[0] : **null**,

    max\_price: priceRange.value[1] !== maxPrice ? priceRange.value[1] : **null**,

    sex: selectedGender.value || **null**,

  }

  // Убираем null значения из объекта фильтров

  const cleanFilters = Object.fromEntries(

    Object.entries(filters).filter(([\_, value]) => value !== **null** && value !== undefined)

  )

  emit('filter-changed', cleanFilters)

}

// Сброс фильтров в начальное состояние

const resetFilters = () => {

  selectedCategory.value = ''

  selectedCountry.value = ''

  selectedGender.value = ''

  priceRange.value = [minPrice, maxPrice]

  emit('filter-changed', {})

}

// Проверка, чтобы минимальная цена не была больше максимальной

const updatePriceText = () => {

  if (priceRange.value[0] > priceRange.value[1]) {

    priceRange.value[0] = priceRange.value[1]

  }

}

onMounted(() => {

  loadFilterData()

})

const sectionClass = 'filter-section-anim' // Класс для анимации

</script>

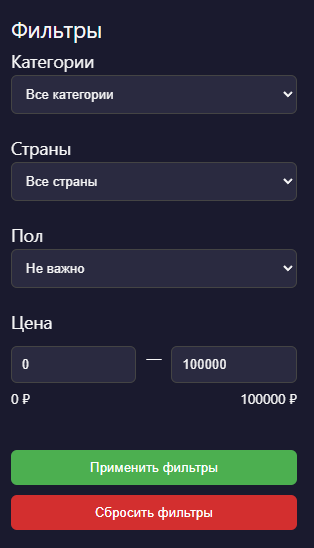
****

Рисунок 19 -Компонент фильтра ProductFilter

**Представление ProductDetailView**

Представление ProductDetailView отвечает за отображение подробной информации о выбранном товаре. Она автоматически открывается при нажатии на карточку товара в компоненте ProductCards. Основная задача — предоставить пользователю расширенную информацию о товаре и отзывы других покупателей.

Страница состоит из двух основных компонентов:

* ProductDetailCard — отображает подробные сведения о товаре: изображение, описание, цену, скидки, характеристики и т.п.
* ProductReviews — показывает отзывы пользователей, а также форму для добавления нового отзыва.

Переход на эту страницу осуществляется программно через router.push() в компоненте ProductCards, используя идентификатор выбранного товара.

**Компонент ProductDetailCard**

Компонент ProductDetailCard отображает подробную информацию о выбранном товаре в интернет-магазине. Он предоставляет пользователю возможность просмотреть основные характеристики товара, выбрать цвет и размер, указать количество и добавить товар в корзину. Компонент также информирует о состоянии загрузки, ошибках и успешных операциях.

Визуальное представление компонента показано на рисунке 20, а полный исходный код приведён в листинге 43.

Основные особенности компонента:

1. Загрузка и отображение товара

* При монтировании компонента происходит асинхронный запрос данных о товаре по его идентификатору.
* Во время загрузки отображается индикатор загрузки.
* В случае ошибки отображается сообщение об ошибке.
* Показывается изображение товара с обработкой ошибок загрузки (отображается запасное изображение, если основное не загрузилось).

1. Отображение информации о товаре

* Выводятся название, описание, категория, страна производства, пол и цена товара.
* Цена форматируется в рублях, пол отображается текстом.

1. Выбор цвета и размера

* Пользователь может выбрать цвет из доступных уникальных вариантов товара.
* После выбора цвета доступные размеры автоматически фильтруются и отображаются.
* Размеры, которых нет в наличии, визуально недоступны для выбора.
* При выборе нового цвета сбрасывается выбор размера.

1. Управление количеством товара

* Можно увеличить или уменьшить количество выбранного товара (минимум 1).
* Количество отображается в поле только для чтения.

1. Добавление товара в корзину

* Кнопка добавления в корзину активна только при выборе цвета, размера и наличии товара.
* При добавлении отправляется запрос на сервер с указанием выбранной комбинации и количеством.
* Отображается уведомление об успешном добавлении товара.
* В случае ошибки отображается сообщение.

**Листинг 43. Компонент ProductDetailCard.vue**

<template>

  <div>

    <!-- Блок загрузки -->

    <div v-if="loading" class="loading-wrapper">

      <div class="spinner"></div>

      <div class="loading-text">Загрузка товара...</div>

    </div>

    <!-- Анимация появления контента после загрузки -->

    <transition name="fade">

      <template v-if="!loading">

        <!-- Основной блок с информацией о товаре -->

        <div v-if="product" class="product-detail">

          <!-- Блок с изображением товара -->

          <div class="image-block">

            <!-- Основное изображение товара -->

            <img

              v-if="imageLoaded"

              :src="getImageUrl(product.photo)"

              :alt="product.name"

              class="main-image"

              @error="handleImageError"

              @load="handleImageLoad"

            />

            <!-- Запасное изображение, если основное не загрузилось -->

            <div v-else class="image-placeholder">

              <img

                :src="placeholderImage"

                :alt="product.name"

                class="placeholder-img"

              />

              <div class="placeholder-overlay">

                <span class="placeholder-text">Фото недоступно</span>

              </div>

            </div>

          </div>

          <!-- Блок с информацией о товаре -->

          <div class="info-block">

            <h1>*{{ product.name }}*</h1>

            <p class="description">*{{ product.description }}*</p>

            <!-- Мета-информация о товаре -->

            <div class="product-meta">

              <p><strong>Категория:</strong> *{{ product.category?.name || 'Неизвестная категория' }}*</p>

              <p><strong>Страна:</strong> *{{ product.country?.name || 'Неизвестная страна' }}*</p>

              <p><strong>Пол:</strong> *{{ formatSex(product.sex) }}*</p>

              <p class="price"><strong>Цена:</strong> *{{ formatPrice(product.price) }}*</p>

            </div>

            <!-- Блок выбора цвета -->

            <div v-if="uniqueColors.length" class="color-section">

              <h3>Доступные цвета:</h3>

              <div class="color-options">

                <div

                  v-for="color in uniqueColors"

                  :key="color.id"

                  class="color-box"

                  :style="{ backgroundColor: color.hex, border: selectedColor?.id === color.id ? '3px solid #ff46a0' : '2px solid #2e2e3f' }"

                  :title="color.name"

                  @click="selectColor(color)"

                ></div>

              </div>

            </div>

            <!-- Блок выбора размера -->

            <div v-if="availableSizes.length" class="size-section">

              <h3>Доступные размеры:</h3>

              <div class="size-options">

                <div

                  v-for="size in availableSizes"

                  :key="size.id"

                  class="size-box"

                  :class="{ selected: selectedSize?.id === size.id, unavailable: !isSizeAvailable(size) }"

                  :title="'Размер: ' + size.name"

                  @click="selectSize(size)"

                >

*{{ size.name }}*

                </div>

              </div>

            </div>

            <!-- Блок добавления в корзину -->

            <div class="add-to-cart-block">

              <div class="quantity-block">

                <label for="quantity">Количество:</label>

                <div class="quantity-controls">

                  <button class="quantity-btn" @click="decreaseQuantity">-</button>

                  <input

                    type="number"

                    id="quantity"

                    v-model="quantity"

                    min="1"

                    class="quantity-input"

                    readonly

                  />

                  <button class="quantity-btn" @click="increaseQuantity">+</button>

                </div>

              </div>

              <button

                class="btn-add-to-cart"

                :disabled="!canAddToCart"

                @click="addToCart"

              >

                Добавить в корзину

              </button>

            </div>

            <!-- Сообщение об ошибке -->

            <p v-if="error" class="error-message">*{{ error }}*</p>

            <!-- Уведомление о добавлении в корзину -->

            <transition name="fade">

              <div v-if="showCartAnimation" class="cart-notification">

                <span>Товар добавлен в корзину</span>

              </div>

            </transition>

          </div>

        </div>

        <!-- Сообщение об ошибке загрузки товара -->

        <div v-else class="error">Ошибка загрузки товара</div>

      </template>

    </transition>

  </div>

</template>

<script setup>

import { ref, onMounted, computed } from 'vue';

import { useRoute } from 'vue-router';

import api from '@/services/api';

import placeholderImage from '@/assets/images/product-placeholder.jpeg';

// Инициализация переменных

const route = useRoute();

const product = ref(null);          // Данные о товаре

const loading = ref(true);          // Флаг загрузки

const productId = route.params.id;  // ID товара из URL

const quantity = ref(1);            // Количество товара

const error = ref('');              // Сообщение об ошибке

const showCartAnimation = ref(false); // Флаг показа анимации добавления в корзину

const imageLoaded = ref(true);      // Флаг загрузки изображения

const selectedColor = ref(null);    // Выбранный цвет

const selectedSize = ref(null);     // Выбранный размер

// Обработчик ошибки загрузки изображения

const handleImageError = () => {

  imageLoaded.value = false;

};

// Обработчик успешной загрузки изображения

const handleImageLoad = () => {

  imageLoaded.value = true;

};

// Загрузка данных о товаре с сервера

const fetchProduct = async () => {

  try {

    const response = await api.get(`/product/${productId}`);

    product.value = response.data;

    imageLoaded.value = true;

  } catch (e) {

    console.error('Ошибка при загрузке данных о товаре:', e);

  } finally {

    loading.value = false;

  }

};

// Вычисляемое свойство для уникальных цветов товара

const uniqueColors = computed(() => {

  const seen = new Set();

  return (product.value?.product\_color\_sizes || [])

    .map(pcs => pcs.color)

    .filter(color => color && !seen.has(color.id) && seen.add(color.id));

});

// Вычисляемое свойство для доступных размеров (в зависимости от выбранного цвета)

const availableSizes = computed(() => {

  if (!selectedColor.value) return [];

  const sizes = product.value.product\_color\_sizes

    .filter(pcs => pcs.color.id === selectedColor.value.id)

    .map(pcs => pcs.size);

  const seen = new Set();

  return sizes.filter(size => size && !seen.has(size.id) && seen.add(size.id));

});

// Проверка доступности размера

const isSizeAvailable = (size) => {

  if (!selectedColor.value) return false;

  return product.value.product\_color\_sizes.some(pcs =>

    pcs.color.id === selectedColor.value.id &&

    pcs.size.id === size.id &&

    pcs.quantity > 0

  );

};

// Проверка возможности добавления в корзину

const canAddToCart = computed(() => {

  return selectedColor.value && selectedSize.value && quantity.value > 0 &&

    isSizeAvailable(selectedSize.value);

});

// Форматирование цены

const formatPrice = (price) => {

  return new Intl.NumberFormat('ru-RU', {

    style: 'currency',

    currency: 'RUB',

    maximumFractionDigits: 0

  }).format(price);

};

// Форматирование пола

const formatSex = (sex) => {

  if (sex === 0) return 'Женский';

  if (sex === 1) return 'Мужской';

  return 'Не указан';

};

// Получение URL изображения

const getImageUrl = (path) => {

  if (!path) {

    imageLoaded.value = false;

    return '';

  }

  return path.startsWith('http') ? path : `http://secrets-of-the-east.ru/storage/${path}`;

};

// Уменьшение количества товара

const decreaseQuantity = () => {

  if (quantity.value > 1) quantity.value--;

};

// Увеличение количества товара

const increaseQuantity = () => {

  quantity.value++;

};

// Выбор цвета

const selectColor = (color) => {

  selectedColor.value = color;

  selectedSize.value = null;

  error.value = '';

};

// Выбор размера

const selectSize = (size) => {

  if (!isSizeAvailable(size)) return;

  selectedSize.value = size;

  error.value = '';

};

// Добавление товара в корзину

const addToCart = async () => {

  error.value = '';

  showCartAnimation.value = false;

  if (!selectedColor.value || !selectedSize.value) {

    error.value = 'Пожалуйста, выберите цвет и размер';

    return;

  }

  // Поиск конкретной комбинации цвета и размера

  const pcs = product.value.product\_color\_sizes.find(pcs =>

    pcs.color.id === selectedColor.value.id &&

    pcs.size.id === selectedSize.value.id

  );

  if (!pcs) {

    error.value = 'Выбранный цвет и размер недоступны';

    return;

  }

  try {

    // Отправка запроса на добавление в корзину

    await api.post(`/cart/product/${pcs.id}`, {

      quantity: quantity.value

    });

    // Показ уведомления об успешном добавлении

    showCartAnimation.value = true;

    setTimeout(() => {

      showCartAnimation.value = false;

    }, 3000);

  } catch (e) {

    error.value = 'Ошибка при добавлении товара в корзину';

    console.error('Ошибка:', e.response?.data?.message || e.message);

  }

};

// Загрузка данных при монтировании компонента

onMounted(fetchProduct);

</script>

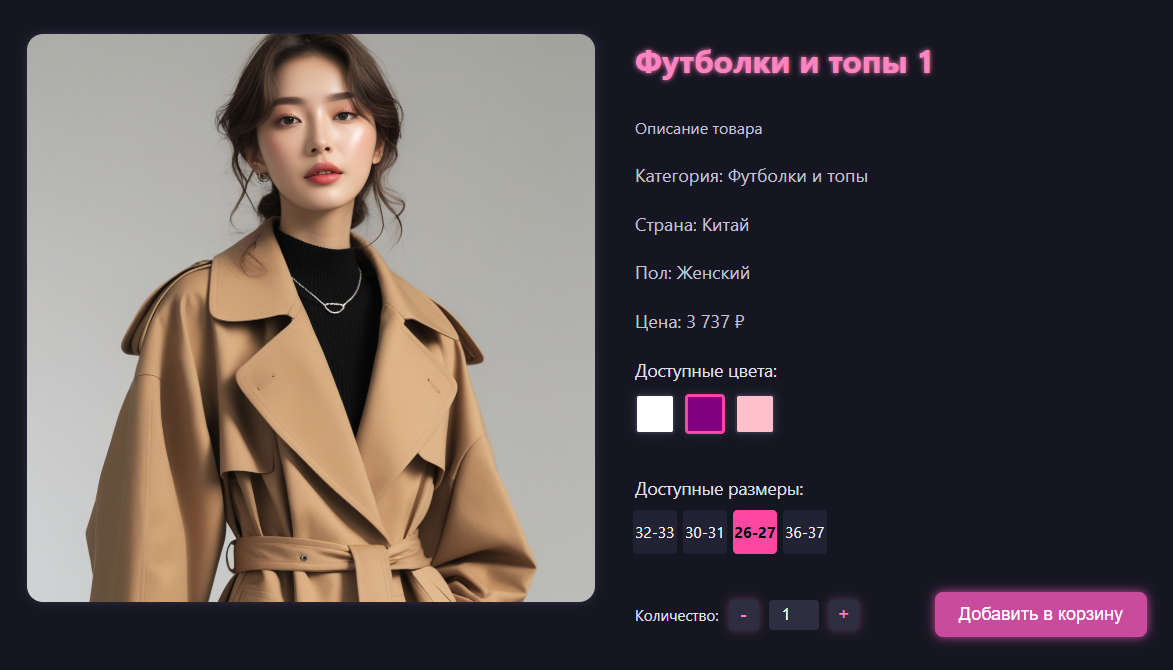
****

Рисунок 20 – Компонент детального товара ProductDetailCard

**Компонент ProductReviews**

Данный компонент отображает отзывы пользователей о конкретном товаре и позволяет авторизованным пользователям, которые уже заказывали этот товар, добавить свой отзыв с рейтингом и комментарием.

В верхней части компонента выводится список оставленных отзывов, где для каждого отображаются имя пользователя, рейтинг в виде звёзд и текст комментария. Если отзывов нет, показывается сообщение об их отсутствии.

Ниже расположена форма для добавления нового отзыва, которая доступна только авторизованным пользователям, ранее совершившим заказ данного товара. В форме пользователь может выбрать рейтинг, кликнув по звёздам, и ввести текст комментария. При попытке отправки происходит проверка заполнения обязательных полей, а также проверка права оставлять отзыв. В случае ошибок запросов или валидации пользователю выводятся информативные сообщения.

Для работы с данными используются асинхронные запросы к API: при монтировании компонента загружаются текущие отзывы с сервера, а новые отзывы отправляются методом POST.

Если пользователь не авторизован или не покупал данный товар, вместо формы отображается сообщение с приглашением авторизоваться или совершить покупку для возможности оставлять отзывы.

Визуальное представление компонента показано на рисунке 21, а полный исходный код приведён в листинге 44.

**Листинг 44. Компонент ProductReviews.vue**

<template>

  <!-- Основной контейнер отзывов -->

  <div class="product-reviews">

    <!-- Блок с отзывами, если они есть -->

    <div v-if="reviews.length">

      <h3 class="reviews-title">Отзывы</h3>

      <div class="reviews-container">

        <!-- Отдельный отзыв -->

        <div v-for="review in reviews" :key="review.id" class="review">

          <div class="review-header">

            <div class="review-user">*{{ review.user.name }}*</div>

            <!-- Рейтинг звездами -->

            <div class="review-stars">

              <span

                v-for="star in 5"

                :key="star"

                class="star"

                :class="{ filled: star <= review.rating }"

              >★</span>

            </div>

          </div>

          <p class="review-text">*{{ review.description }}*</p>

          <!-- Кнопка удаления (видна админу/менеджеру/автору) -->

          <button

            v-if="canDeleteReview(review)"

            @click="deleteReview(review.id)"

            class="delete-button"

          >

            Удалить

          </button>

        </div>

      </div>

    </div>

    <!-- Блок, когда отзывов нет -->

    <div v-else class="no-reviews">

      <div class="no-reviews-content">

        <h3>Нет отзывов для этого товара</h3>

      </div>

    </div>

    <!-- Форма добавления нового отзыва -->

    <div v-if="isLoggedIn" class="review-form">

      <h3>Оставьте свой отзыв</h3>

      <form @submit.prevent="submitReview">

        <!-- Поле оценки -->

        <div class="form-group rating-group">

          <label>Оценка:</label>

          <div class="rating-stars">

            <span

              v-for="star in 5"

              :key="star"

              class="star"

              :class="{ selected: star <= newReview.rating }"

              @click="newReview.rating = star"

              role="button"

              tabindex="0"

            >★</span>

          </div>

        </div>

        <!-- Поле текста отзыва -->

        <div class="form-group">

          <label for="description">Комментарий:</label>

          <textarea

            v-model="newReview.description"

            id="description"

            rows="4"

            required

            placeholder="Напишите ваш отзыв здесь..."

          ></textarea>

        </div>

        <!-- Сообщение об ошибке -->

        <p v-if="errorMessage" class="error-message">*{{ errorMessage }}*</p>

        <button type="submit">Отправить отзыв</button>

      </form>

    </div>

    <!-- Блок для неавторизованных пользователей -->

    <div v-else class="login-warning">

      <p class="warning-text">Чтобы оставить отзыв, необходимо авторизоваться через профиль в шапке профиля.</p>

    </div>

  </div>

</template>

<script setup>

import { ref, computed, onMounted } from 'vue';

import { useStore } from 'vuex';

import { useRoute } from 'vue-router';

import api from '@/services/api';

const store = useStore();

const route = useRoute();

const productId = route.params.id;

// Реактивные данные

const reviews = ref([]);

const newReview = ref({ rating: 0, description: '' });

const errorMessage = ref('');

// Вычисляемые свойства из хранилища

const isLoggedIn = computed(() => store.state.user.loggedIn);

const currentUserEmail = computed(() => store.state.user.email);

const userRole = computed(() => store.state.user.role);

// Загрузка отзывов для товара

const fetchReviews = async () => {

  try {

    const response = await api.get(`/product/${productId}/review`);

    reviews.value = response.data;

  } catch (error) {

    console.error('Ошибка загрузки отзывов:', error);

  }

};

// Отправка нового отзыва

const submitReview = async () => {

  errorMessage.value = '';

  // Валидация перед отправкой

  if (!newReview.value.rating || !newReview.value.description.trim()) {

    errorMessage.value = 'Пожалуйста, заполните оценку и комментарий.';

    return;

  }

  try {

    const response = await api.post(`/product/${productId}/review`, newReview.value);

    reviews.value.push(response.data);

    newReview.value = { rating: 0, description: '' }; // Сброс формы

  } catch (error) {

    // Обработка ошибок с сервера

    if (error.response?.data?.message) {

      errorMessage.value = error.response.data.message;

    } else if (error.response?.data?.errors) {

      errorMessage.value = Object.values(error.response.data.errors).flat().join(' ');

    } else {

      errorMessage.value = 'Произошла ошибка при отправке отзыва.';

    }

  }

};

// Проверка прав на удаление отзыва

const canDeleteReview = (review) => {

  const isOwner = review.user?.email === currentUserEmail.value;

  const isAdminOrManager = userRole.value === 1 || userRole.value === 2;

  return isOwner || isAdminOrManager;

};

// Удаление отзыва

const deleteReview = async (reviewId) => {

  try {

    await api.delete(`/product/${productId}/review/${reviewId}`);

    reviews.value = reviews.value.filter((review) => review.id !== reviewId);

  } catch (error) {

    console.error('Ошибка при удалении отзыва:', error);

    errorMessage.value = 'Не удалось удалить отзыв';

  }

};

// Загрузка отзывов при монтировании компонента

onMounted(fetchReviews);

</script>

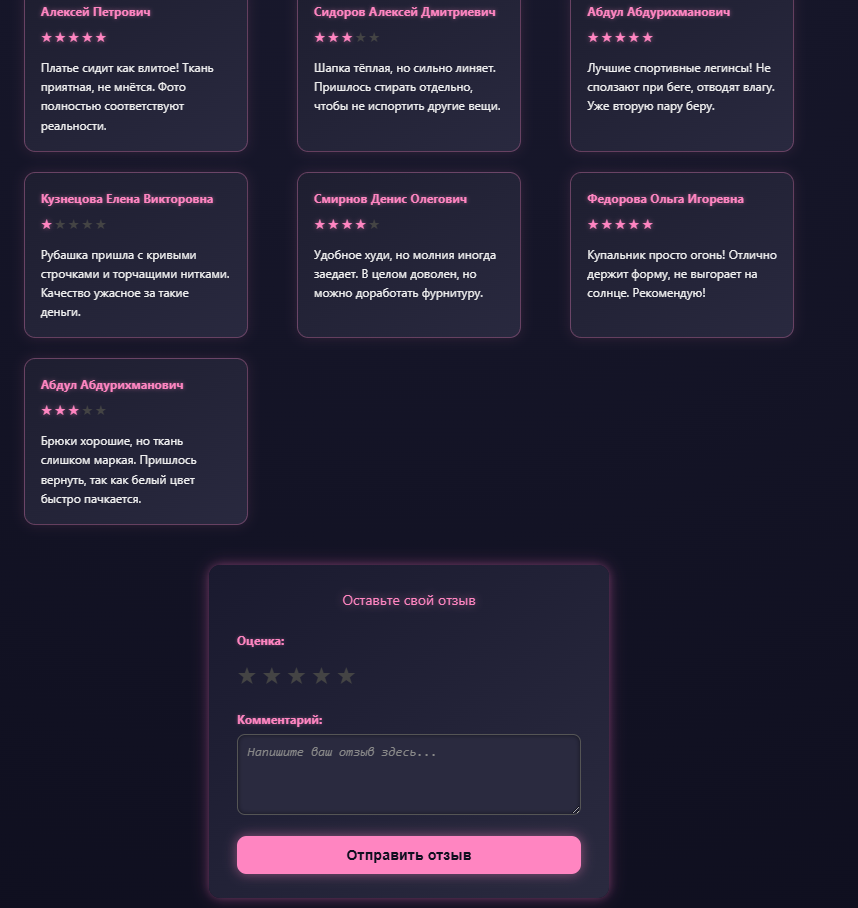
****

Рисунок 21 – Компонент отзывов ProductReviews

**Представление CartView**

Компонент представления корзины разделён на две части: слева отображается список товаров в корзине, справа — блок оформления заказа с итоговой суммой.

В левой части используется дочерний компонент CartItems, которому передаётся массив продуктов из корзины. Компонент CartItems отвечает за визуализацию каждого товара, а также уведомляет родителя о необходимости обновить данные корзины через событие update.

Справа размещён компонент CartCheckout, который получает общую стоимость корзины и отображает блок с кнопками для оформления заказа.

Основные данные корзины (products и total\_cost) загружаются с сервера при монтировании компонента функцией fetchCart. Она делает запрос к API и обновляет состояние cart. При ошибках загрузки в консоль выводятся сообщения. Полный код страницы приведён в листинге 45.

**Листинг 45. Представление CartView.vue**

<template>

  <div class="cart-container">

    <!-- Левая часть: список товаров -->

    <div class="left-side">

      <CartItems :products="cart.products" @update="fetchCart" />

    </div>

    <!-- Правая часть: оформление заказа -->

    <div class="right-side">

      <CartCheckout :totalCost="cart.total\_cost" />

    </div>

  </div>

</template>

<script setup>

import { ref, onMounted } from "vue";

import api from "@/services/api";

import CartItems from '@/components/cart/CartItems.vue'

import CartCheckout from '@/components/cart/CartCheckout.vue'

const cart = ref({

  products: [],

  total\_cost: 0,

});

// Загрузка корзины с сервера

async function fetchCart() {

  try {

    const response = await api.get("/cart");

    if (response.data.products && Array.isArray(response.data.products)) {

      cart.value = response.data;

    } else {

      console.error("В ответе API нет массива products");

    }

  } catch (error) {

    console.error("Ошибка загрузки корзины:", error);

  }

}

onMounted(() => {

  fetchCart();

});

</script>

**Компонент CartItems**

Компонент CartItems реализует функциональность отображения и управления товарами в корзине интернет-магазина. Он отвечает за загрузку содержимого корзины с сервера, визуальное представление каждого товара и интерактивное изменение количества или удаление позиций.

При открытии компонента запускается асинхронная загрузка корзины через API. Пока данные загружаются, отображается индикатор загрузки. Если корзина пуста, пользователю показывается соответствующее сообщение.

Каждый товар представлен в виде карточки с изображением (или заглушкой при отсутствии изображения), названием, характеристиками (цвет, размер), ценой за единицу и общей стоимостью в зависимости от количества. Пользователь может изменять количество с помощью кнопок «−» и «+». Кнопка уменьшения блокируется при достижении минимального значения (1), чтобы избежать недопустимого количества. Также предусмотрена кнопка удаления товара из корзины.

Все изменения количества и удаления отправляются на сервер и при успешном ответе немедленно обновляют локальное состояние, обеспечивая мгновенную реакцию интерфейса без повторной полной перезагрузки корзины.

Основные особенности компонента:

1. Загрузка корзины

* При монтировании компонента отправляется запрос /cart для получения списка товаров.
* Если запрос успешен, товары сохраняются в локальное состояние products.
* При ошибке загрузки корзина отображается как пустая.

1. Отображение карточек товаров

* Для каждого товара отображается изображение, или заглушка при ошибке загрузки.
* Показаны цвет, размер, цена за единицу и итоговая стоимость.

1. Изменение количества

* Кнопки увеличения и уменьшения позволяют регулировать количество товара.
* Кнопка уменьшения становится неактивной, если количество товара — 1.
* При каждом изменении выполняется PATCH-запрос на сервер и обновляется локальное количество.

1. Удаление товара

* Кнопка удаления вызывает DELETE-запрос на API, после чего товар удаляется из отображения.

1. Обработка изображений

* При ошибке загрузки изображения автоматически отображается стандартная заглушка.

Визуальное представление компонента показано на рисунке 22, а полный исходный код компонента приведён в листинге 46.

**Листинг 46. Компонент CartItems.vue**

<template>

  <div class="cart-items-container">

    <!-- Состояние загрузки -->

    <div v-if="loading" class="loading">Загрузка...</div>

    <!-- Основное содержимое, если не загружается -->

    <div v-else>

      <!-- Если корзина пуста -->

      <div v-if="products.length === 0" class="empty">Корзина пуста</div>

      <!-- Список товаров в корзине -->

      <div class="products-list">

        <div

          v-for="product in products"

          :key="product.productColorSizeId"

          class="product-card"

          tabindex="0"

          aria-label="Товар в корзине"

        >

          <!-- Изображение товара или заглушка -->

          <div class="product-image-container">

            <!-- Если изображение загрузилось — показываем его -->

            <img

              v-if="product.imageLoaded"

              :src="product.photo"

              alt="Фото товара"

              class="product-photo"

              @error="() => handleImageError(product)"

              @load="() => handleImageLoad(product)"

            />

            <!-- Если изображение не загрузилось — показываем заглушку -->

            <div v-else class="image-placeholder">

              <img

                src="@/assets/images/product-placeholder.jpeg"

                alt="Фото недоступно"

                class="placeholder-img"

              />

              <div class="placeholder-overlay">

                <span class="placeholder-text">Фото недоступно</span>

              </div>

            </div>

          </div>

          <!-- Информация о товаре -->

          <div class="product-info">

            <h3 class="product-name">*{{ product.name }}*</h3>

            <p>

              Цвет: <span class="highlight">*{{ product.color || "—" }}*</span>,

              Размер: <span class="highlight">*{{ product.size || "—" }}*</span>

            </p>

            <p>Цена за шт: <span class="price">*{{ product.price }}* ₽</span></p>

            <p>Итого: <span class="price-total">*{{ product.price \* product.quantity }}* ₽</span></p>

          </div>

          <!-- Управление количеством и кнопка удаления -->

          <div class="product-controls">

            <div class="quantity-controls" role="group" aria-label="Управление количеством">

              <!-- Кнопка уменьшения количества -->

              <button

                @click="updateQuantity(product, product.quantity - 1)"

                :disabled="product.quantity <= 1"

                aria-disabled="product.quantity <= 1"

                aria-label="Уменьшить количество"

              >

                -

              </button>

              <!-- Отображение текущего количества -->

              <span class="quantity" aria-live="polite">*{{ product.quantity }}*</span>

              <!-- Кнопка увеличения количества -->

              <button

                @click="updateQuantity(product, product.quantity + 1)"

                aria-label="Увеличить количество"

              >

                +

              </button>

            </div>

            <!-- Кнопка удаления товара из корзины -->

            <button class="remove-btn" @click="removeProduct(product)" aria-label="Удалить товар">

              Удалить

            </button>

          </div>

        </div>

      </div>

    </div>

  </div>

</template>

<script setup>

import { ref, onMounted } from "vue";

import api from "@/services/api";

// Список товаров в корзине

const products = ref([]);

// Состояние загрузки

const loading = ref(false);

// Обработчик ошибки загрузки изображения — отображаем заглушку

const handleImageError = (product) => {

  product.imageLoaded = false;

};

// Обработчик успешной загрузки изображения

const handleImageLoad = (product) => {

  product.imageLoaded = true;

};

// Загрузка корзины при открытии компонента

async function loadCart() {

  loading.value = true;

  try {

    // Получаем список товаров из API

    const response = await api.get("/cart");

    products.value = response.data.products.map((p) => ({

      ...p,

      // Используем productColorSizeId как уникальный ключ (если есть)

      productColorSizeId: p.productColorSizeId || p.id,

      imageLoaded: **true**, // Сначала предполагаем, что изображение загрузится

    }));

  } catch (error) {

    console.error("Ошибка загрузки корзины:", error);

    products.value = []; // В случае ошибки — очищаем корзину

  } finally {

    loading.value = false;

  }

}

// Обновление количества товара

async function updateQuantity(product, newQuantity) {

  if (newQuantity < 1) return;

  try {

    // Отправляем новое количество на сервер

    await api.patch(`/cart/product/${product.productColorSizeId}`, {

      quantity: newQuantity,

    });

    // Обновляем локально

    product.quantity = newQuantity;

  } catch (error) {

    console.error("Ошибка обновления количества:", error);

  }

}

// Удаление товара из корзины

async function removeProduct(product) {

  try {

    // Удаляем товар через API

    await api.delete(`/cart/product/${product.productColorSizeId}`);

    // Удаляем из локального списка

    products.value = products.value.filter(

      (p) => p.productColorSizeId !== product.productColorSizeId

    );

  } catch (error) {

    console.error("Ошибка удаления товара:", error);

  }

}

// Загружаем корзину при монтировании компонента

onMounted(() => {

  loadCart();

});

</script>



Рисунок 22 – Компоненты CartItems и CartCheckout

**Компонент CartCheckout**

Компонент CartCheckout реализует завершающий этап процесса покупки — оформление заказа. Он включает выбор пункта выдачи, подтверждение общей стоимости и переход к оплате, если она необходима. Поведение компонента тесно связано с глобальным состоянием vuex, в частности — с выбранным пунктом выдачи.

1. Выбор пункта выдачи

В верхней части интерфейса отображается информация о текущем выбранном пункте выдачи (город, улица, дом, дополнительная информация). Пользователь может изменить его, вызвав модальное окно выбора нового адреса.

Если адрес не выбран, кнопка оформления не доступна пока не будет выбран пункт выдачи.

1. Информация об итоговой стоимости

Под адресом отображается сумма заказа — totalcost. Если сумма равна нулю, либо не выбран пункт выдачи, либо идёт процесс оформления, кнопка оформления заказа становится неактивной, предотвращая ошибочные действия.

1. Процесс оформления заказа

При нажатии на кнопку «оформить заказ» вызывается метод placeorder, который:

Проверяет, выбран ли пункт выдачи. Если нет — кнопка оформления не доступна пока не будет выбран пункт выдачи.

Отправляет post-запрос на /order с point\_id выбранного адреса.

Если сервер возвращает status: "payment\_required", происходит перенаправление на тестовую платёжную страницу.

1. Обработка результата оплаты

После возврата с платёжной страницы, в url присутствует параметр success=true или success=false, который автоматически обрабатывается в onmounted():

Success=true — отображается сообщение об успешной оплате и оформлении заказа.

Success=false — пользователь отменил платёж, и заказ не был завершён.

После этого параметр success удаляется из url, чтобы исключить повторное отображение при обновлении страницы.

1. Модальное окно уведомлений

Компонент содержит встроенное модальное окно (modal), которое отображает результат операций: успех, ошибку или отмену. Оно включается при любом важном событии — как при ошибке выбора адреса, так и после завершения оформления заказа.

Таким образом, компонент CartCheckout замыкает весь цикл оформления заказа: от подтверждения адреса до перенаправления на оплату. Это обеспечивает плавный и понятный пользовательский опыт.

Визуальное отображение компонента представлено на рисунке 22, а его полный исходный код приведён в листинге 47.

**Листинг 47. Компонент CartCheckout.vue**

<template>

  <div class="cart-checkout">

    <!-- Если выбран пункт выдачи -->

    <div v-if="selectedPoint" class="delivery-point-info">

      <h4>Пункт выдачи:</h4>

      <p>*{{ selectedPoint.city }}*, *{{ selectedPoint.street }}*, *{{ selectedPoint.house }}*</p>

      <p v-if="selectedPoint.additional\_info">(*{{ selectedPoint.additional\_info }}*)</p>

      <button class="change-point-btn" @click="changePoint">Изменить</button>

    </div>

    <!-- Если не выбран -->

    <div v-else class="no-point-selected">

      <p>Пункт выдачи не выбран</p>

      <button class="select-point-btn" @click="selectPoint">Выбрать пункт</button>

    </div>

    <!-- Итого и кнопка -->

    <h3>Итого</h3>

    <p class="total-cost">*{{ totalCost }}* ₽</p>

    <button

      class="order-btn"

      @click="placeOrder"

      :disabled="totalCost === 0 || !selectedPoint || isLoading"

    >

      {{ isLoading ? "Оформляем..." : "Оформить заказ" }}

    </button>

    <!-- Модалка -->

    <div v-if="modal.visible" class="modal-overlay" @click.self="modal.visible = false">

      <div class="modal-content">

        <h4>*{{ modal.title }}*</h4>

        <p>*{{ modal.message }}*</p>

        <button @click="modal.visible = false">Закрыть</button>

      </div>

    </div>

  </div>

</template>

<script setup>

import {computed, ref, onMounted} from "vue";

import {useStore} from "vuex";

import {useRoute, useRouter} from "vue-router";

import api from "@/services/api";

const props = defineProps({

  totalCost: {

    type: Number,

    default: 0,

  },

});

const store = useStore();

const route = useRoute();

const router = useRouter();

const isLoading = ref(false);

// Модальное окно для любых уведомлений

const modal = ref({

  visible: **false**,

  title: "",

  message: ""

});

// Получаем выбранный пункт

const selectedPoint = computed(() => store.state.selectedPoint);

// Выбрать/изменить пункт

const selectPoint = () => store.commit("setShowDeliveryModal", true);

const changePoint = () => store.commit("setShowDeliveryModal", true);

// Обработка оплаты Stripe через параметры success в URL

onMounted(() => {

  const success = route.query.success;

  if (success === "true") {

    showModal("Успешно", "Оплата прошла успешно. Заказ оформлен!");

  } else if (success === "false") {

    showModal("Отмена", "Вы отменили оплату. Заказ не был оформлен.");

  }

  // Удаляем параметр из URL

  if (success !== undefined) {

    const newQuery = {...route.query};

    delete newQuery.success;

    router.replace({query: newQuery});

  }

});

// Отправка заказа

const placeOrder = async () => {

  if (!selectedPoint.value) {

    showModal("Ошибка", "Пожалуйста, выберите пункт выдачи.");

    return;

  }

  isLoading.value = true;

  try {

    const response = await api.post("/order", {

      point\_id: selectedPoint.value.id,

    });

    if (response.data.status === "payment\_required") {

      // Переход на Stripe

      window.location.href = response.data.payment\_url;

    } else {

      showModal("Успешно", "Заказ успешно оформлен без оплаты.");

    }

  } catch (error) {

    const msg = error.response?.data?.message || "Ошибка при оформлении заказа.";

    showModal("Ошибка", msg);

  } finally {

    isLoading.value = false;

  }

};

// Показ уведомления

function showModal(title, message) {

  modal.value.title = title;

  modal.value.message = message;

  modal.value.visible = true;

}

</script>

**Компонент модального окна выбора пункта выдачи (PointsModal)**

Компонент PointsModal реализует интерфейс выбора пункта самовывоза, необходимого для оформления заказа. Он представлен в виде модального окна, в котором отображается список доступных пунктов выдачи, полученных с сервера через API.

Основной функционал:

* При открытии модального окна происходит загрузка данных с /point. Во время загрузки отображается соответствующее сообщение, а в случае ошибки выводится сообщение об ошибке.
* Пользователь может выбрать любой из доступных пунктов, после чего:

1. Выбранный пункт сохраняется в хранилище Vuex.
2. Также дублируется в localStorage для восстановления при перезагрузке страницы.
3. Модальное окно закрывается автоматически.

* Компонент поддерживает восстановление ранее выбранного пункта, если он уже был сохранён.

Поведение при открытии:

Каждое открытие окна сопровождается повторной попыткой загрузки данных, чтобы обеспечить актуальность информации о доступных пунктах.

Визуальное отображение интерфейса компонента представлено на рисунке 23, а полный исходный код в листинге 48.

**Листинг 48. Компонент PointsModal.vue**

<template>

  <div class="modal-overlay" v-show="show" @click.self="close">

    <div class="modal-content">

      <button class="close-button" @click="close">×</button>

      <h2 class="modal-title">Пункты выдачи</h2>

      <div class="points-list">

        <!-- Состояния загрузки -->

        <div v-if="loading" class="loading">Загрузка...</div>

        <div v-else-if="error" class="error">*{{ error }}*</div>

        <div v-else-if="points.length === 0" class="no-points">Нет доступных пунктов выдачи</div>

        <!-- Список пунктов выдачи -->

        <div

          v-else

          v-for="point in points"

          :key="point.id"

          class="point-item"

          :class="{ selected: selectedPoint && selectedPoint.id === point.id }"

          @click="selectPoint(point)"

          tabindex="0"

          role="button"

        >

          <div class="point-card">

            <div class="point-header">

              <h3>*{{ point.city }}*</h3>

            </div>

            <div class="point-details">

              <p>Адрес: *{{ point.street }}*, *{{ point.house }}*</p>

              <p v-if="point.additional\_info" class="additional-info">Дополнительно: *{{ point.additional\_info }}*</p>

            </div>

          </div>

        </div>

      </div>

    </div>

  </div>

</template>

<script>

import api from '@/services/api';

export default {

  name: 'PointsModal',

  props: {

    show: Boolean

  },

  data() {

    return {

      points: [],         // список всех пунктов выдачи

      loading: **false**,     // индикатор загрузки пунктов

      error: **null**,        // ошибка загрузки

      selectedPoint: **null** // текущий выбранный пункт (локально в компоненте)

    };

  },

  methods: {

    // Закрытие модального окна

    close() {

      this.$emit('close');

    },

    // Загрузка списка пунктов с API

    async fetchPoints() {

      try {

        this.loading = true;

        this.error = null;

        const response = await api.get('/point');

        this.points = response.data;

        // Если уже есть выбранный пункт в Vuex или localStorage, восстанавливаем его

        const savedPoint = this.$store.getters.selectedPoint;

        if (savedPoint) {

          this.selectedPoint = savedPoint;

        } else {

          // Если в хранилище нет, попробуем из localStorage (на случай если Vuex ещё не инициализирован)

          const localPoint = localStorage.getItem('selectedPoint');

          if (localPoint) {

            try {

              this.selectedPoint = JSON.parse(localPoint);

            } catch {

              this.selectedPoint = null;

            }

          }

        }

      } catch (err) {

        this.error = 'Не удалось загрузить пункты выдачи';

      } finally {

        this.loading = false;

      }

    },

    // Обработка выбора пункта

    selectPoint(point) {

      this.selectedPoint = point;

      // Сохраняем выбранный пункт в Vuex

      this.$store.commit("setSelectedPoint", point);

      // Сохраняем в localStorage (на всякий случай)

      localStorage.setItem("selectedPoint", JSON.stringify(point));

      // Передаём выбор родителю, если нужно

      this.$emit("select", point);

      // Закрываем модалку

      this.close();

    }

  },

  mounted() {

    this.fetchPoints();

  },

  watch: {

    // При открытии модалки повторно загружаем пункты и сбрасываем выбранный пункт

    show(newVal) {

      if (newVal) {

        // Не сбрасываем selectedPoint при открытии, чтобы всегда показывался последний выбранный

        this.fetchPoints();

      }

    }

  }

};

</script>

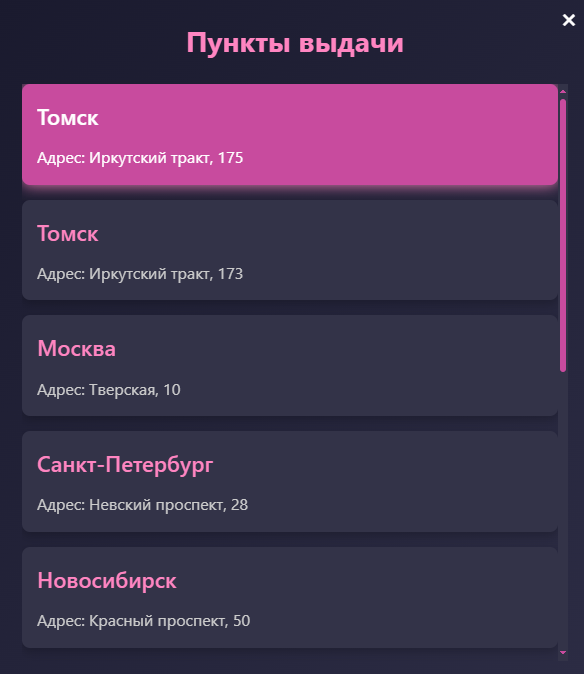
****

Рисунок 23 – Компонент PointsModal

**Компонент модального окна просмотра заказов (MyOrderModal)**

Компонент MyOrderModal предоставляет пользователю интерфейс для просмотра оформленных заказов, а также для детального ознакомления с содержимым каждого заказа и возможностью его отмены. Компонент реализован в виде модального окна с двумя вкладками: список заказов и детали конкретного заказа.

Основной функционал:

1. Список заказов:

* Загружается автоматически при открытии компонента.
* Отображается базовая информация: номер заказа, дата, адрес доставки, сумма и статус.
* По клику на заказ загружаются его подробности.

1. Детали заказа:

* Показываются товары, входящие в заказ, включая название, изображение, количество, цвет и размер.
* Также отображается общая сумма, адрес и текущий статус заказа.

1. Отмена заказа:

* Пользователь может отменить заказ, если он ещё не выполнен или не отменён ранее.
* Запрос на отмену отправляется через API, после чего статус заказа обновляется.

Полный код компонента представлен в **листинге 49**, а внешний вид интерфейса показан на **рисунке 24**.

**Листинг 49. Компонент MyOrderModal.vue**

<template>

  <!-- Оверлей модалки, клик по нему закрывает окно -->

  <div class="modal-overlay" @click.self="closeModal">

    <div class="modal-content">

      <h2 class="modal-title">Мои заказы</h2>

      <!-- Вкладки: список заказов и детали заказа -->

      <div class="tabs">

        <!-- Кнопка для списка заказов -->

        <button :class="{ active: tab === 'list' }" @click="tab = 'list'">

          Список заказов

        </button>

        <!-- Кнопка для деталей заказа отображается, если выбран заказ -->

        <button v-if="selectedOrder" :class="{ active: tab === 'details' }" @click="tab = 'details'">

          Детали заказа

        </button>

      </div>

      <!-- Вкладка "Список заказов" -->

      <div v-if="tab === 'list'">

        <div v-if="loading">Загрузка...</div>

        <div v-else-if="!orders.length">Нет заказов</div>

        <ul v-else class="order-list">

          <!-- Список заказов -->

          <li

            v-for="order in orders"

            :key="order.id"

            class="order-item"

            @click="loadOrderDetails(order.id)"

          >

            <strong>Заказ №*{{ order.id }}*</strong><br />

            <span>Статус: <strong>*{{ order.status }}*</strong></span><br />

            Дата: *{{ formatDate(order.order\_date) }}*<br />

            Адрес: *{{ formatAddress(order.address) }}*<br />

            Сумма: *{{ order.total\_cost }}* ₽

          </li>

        </ul>

      </div>

      <!-- Вкладка "Детали заказа" -->

      <div v-if="tab === 'details' && selectedOrder" class="order-details">

        <h3>Заказ №*{{ selectedOrder.id }}*</h3>

        <p>Дата: *{{ formatDate(selectedOrder.order\_date) }}*</p>

        <p>Адрес: *{{ formatAddress(selectedOrder.address) }}*</p>

        <p>Итого: *{{ selectedOrder.total\_cost }}* ₽</p>

        <p>Статус: <strong>*{{ selectedOrder.status }}*</strong></p>

        <h4>Товары:</h4>

        <ul class="items-list">

          <li v-for="item in selectedOrder.items" :key="item.product\_id" class="item-card">

            <img :src="item.photo" alt="Фото товара" />

            <div class="item-info">

              <strong class="item-name">*{{ item.name }}*</strong>

              <div class="item-quantity-price">

*{{ item.quantity }}* шт

              </div>

              <div class="item-params">

                Цвет: <span>*{{ item.color || '—' }}*</span>, Размер: <span>*{{ item.size || '—' }}*</span>

              </div>

            </div>

          </li>

        </ul>

        <!-- Кнопка для отмены заказа -->

        <button

          @click="cancelOrder"

          class="cancel-btn"

          :disabled="cancelLoading"

          v-if="selectedOrder.status !== 'Отменён' && selectedOrder.status !== 'Выполнен'"

        >

          {{ cancelLoading ? 'Отмена...' : 'Отменить заказ' }}

        </button>

      </div>

      <!-- Кнопка закрытия модалки -->

      <button class="close-modal-btn" @click="closeModal">Закрыть</button>

    </div>

  </div>

</template>

<script setup>

import { ref, onMounted } from 'vue'

import api from '@/services/api'

// Событие для закрытия модалки

const emit = defineEmits(['close'])

const tab = ref('list')

const orders = ref([])

const selectedOrder = ref(null)

const loading = ref(false)

const cancelLoading = ref(false)

// Закрыть модалку

function closeModal() {

  emit('close')

}

// Загрузить список заказов с сервера

async function fetchOrders() {

  loading.value = **true**

  try {

    const res = await api.get('/order')

    // Если данные в data.data или в data

    orders.value = Array.isArray(res.data) ? res.data : res.data.data || []

  } catch (e) {

    alert('Ошибка загрузки заказов')

    console.error(e)

  } finally {

    loading.value = **false**

  }

}

// Загрузить детали заказа по id

async function loadOrderDetails(id) {

  loading.value = **true**

  try {

    const res = await api.get(`/order/${id}`)

    selectedOrder.value = res.data

    tab.value = 'details'

  } catch (e) {

    alert('Ошибка при загрузке деталей заказа')

    console.error(e)

  } finally {

    loading.value = **false**

  }

}

// Отмена заказа с запросом на сервер

async function cancelOrder() {

  if (!selectedOrder.value) return;

  if (!confirm('Вы уверены, что хотите отменить заказ?')) return;

  cancelLoading.value = true;

  try {

    // Отправляем PATCH запрос на правильный URL для отмены

    await api.patch(`/order/cancelled/${selectedOrder.value.id}`);

    alert('Заказ отменён');

    await fetchOrders();

    selectedOrder.value = null;

    tab.value = 'list';

  } catch (e) {

    alert('Ошибка при отмене заказа');

    console.error(e);

  } finally {

    cancelLoading.value = false;

  }

}

// Форматирование даты

function formatDate(dateStr) {

  if (!dateStr) return ''

  const d = new Date(dateStr)

  return d.toLocaleDateString('ru-RU', {

    day: '2-digit', month: '2-digit', year: 'numeric'

  }) + ' ' + d.toLocaleTimeString('ru-RU', {

    hour: '2-digit', minute: '2-digit'

  })

}

// Форматирование адреса

function formatAddress(addr) {

  if (!addr) return '—'

  if (typeof addr === 'string') return addr

  return [addr.city, addr.street, addr.house].filter(Boolean).join(', ')

}

// При монтировании загружаем список заказов

onMounted(() => {

  fetchOrders()

})

</script>

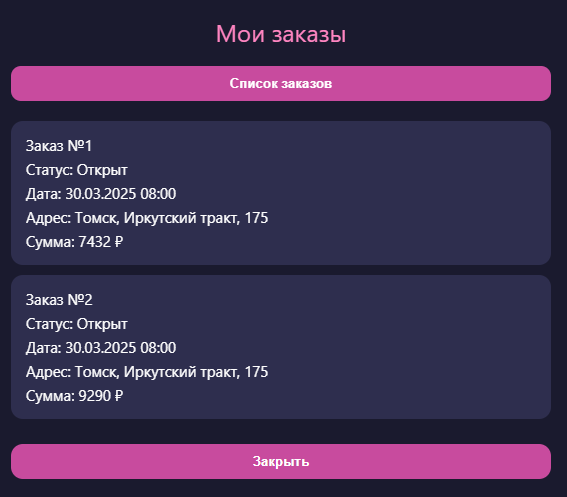
****

Рисунок 24 – Компонент MyOrderModal

**Компонент административной панели (AdminPanelModal)**

Компонент AdminPanelModal представляет собой основное модальное окно управления сайтом для пользователей с правами администратора или менеджера. Это центральный элемент административного интерфейса, через который осуществляется доступ ко всем функциям управления контентом и пользователями.

Основной функционал:

1. Ролевая проверка доступа:

Панель отображает элементы в зависимости от роли пользователя:

* Менеджер (role = 2) видит доступ к товарам и заказам.
* Администратор (role = 1) имеет полный доступ, включая управление категориями, пунктами выдачи, менеджерами и всеми пользователями.

1. Навигация по подмодулям:

Все действия реализованы через вложенные модальные окна, каждое из которых отвечает за определённый раздел:

* CreateProductModal — создание нового товара.
* UserOrdersModal — просмотр и обработка заказов.
* CreateManagerModal — управление менеджерами.
* CreateCategoryModal — добавление/удаление категорий.
* CreatePointsModal — управление пунктами выдачи.
* AllUsersModal — отображение списка всех зарегистрированных пользователей.

Логика взаимодействия:

* Переменные состояния (ref) определяют, какая из модалок активна в текущий момент.
* Кнопки управления запускают соответствующие модальные окна.
* Повторное открытие UserOrdersModal реализовано через nextTick, что позволяет перезапустить компонент для обновления данных (например, после закрытия и повторного открытия).

Внешний вид интерфейса показан на **рисунке 25**.

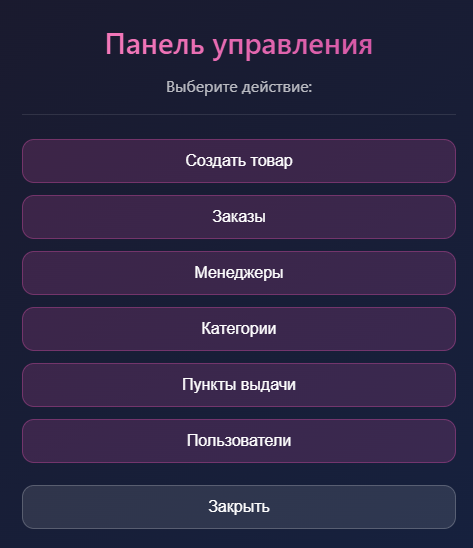


Рисунок 25 – Компонент AdminPanelModal

**Компонент AllUsersModal**

Компонент AllUsersModal реализует модальное окно для администраторов, позволяющее просматривать список всех пользователей и редактировать данные выбранного пользователя.

1. Список пользователей

При открытии отправляется запрос на /user, после чего пользователи отображаются в виде списка. При клике по пользователю открывается вкладка с его данными.

1. Редактирование пользователя

Вкладка профиля позволяет изменить имя, email, телефон и пол пользователя. Изменения отправляются на сервер с помощью PATCH-запроса. После успешного обновления список перезагружается.

1. Переключение вкладок и закрытие

Реализованы две вкладки: «Список» и «Профиль». Переход между ними зависит от выбора пользователя. Компонент можно закрыть кнопкой или кликом по фону.

Компонент обеспечивает базовую административную функциональность для управления пользователями. Визуальное представление — на рисунке 26, исходный код — в листинге 50.

**Листинг 50. Компонент AllUsersModal.vue**

  <template>

    <div class="modal-overlay" @click.self="emit('close')">

      <div class="modal-content">

        <h2 class="modal-title">Пользователи</h2>

        <!-- Вкладки переключения между списком и профилем -->

        <div class="tabs">

          <!-- Активная вкладка подсвечивается -->

          <button :class="{active: tab === 'list'}" @click="tab = 'list'">Список</button>

          <!-- Вкладка профиля доступна только если выбран пользователь -->

          <button :class="{active: tab === 'detail'}" :disabled="!selectedUser" @click="tab = 'detail'">Профиль</button>

        </div>

        <!-- Отображение списка пользователей -->

        <div v-if="tab === 'list'">

          <!-- Показ индикатора загрузки -->

          <div v-if="loading">Загрузка...</div>

          <ul v-else class="manager-list">

            <!-- Перебор массива пользователей, клик выбирает пользователя -->

            <li v-for="user in users" :key="user.id" class="manager-item" @click="selectUser(user)">

              <div>

                <strong>*{{ user.name }}*</strong><br />

*{{ user.email }}*<br />

                Роль: *{{ user.role.name }}*

              </div>

            </li>

          </ul>

        </div>

        <!-- Отображение профиля выбранного пользователя -->

        <div v-if="tab === 'detail' && selectedUser">

          <!-- Поля для редактирования -->

          <input v-model="selectedUser.name" class="modal-input" placeholder="Имя" />

          <input v-model="selectedUser.email" class="modal-input" placeholder="Email" />

          <input v-model="selectedUser.telephone" class="modal-input" placeholder="Телефон" />

          <select v-model="selectedUser.sex" class="modal-input">

            <option disabled value="">Пол</option>

            <option :value="0">Мужской</option>

            <option :value="1">Женский</option>

          </select>

          <!-- Кнопка сохранения изменений -->

          <button class="submit-btn" @click="updateUser">Сохранить</button>

          <!-- Кнопка возврата к списку -->

          <button class="cancel-btn" @click="tab = 'list'">Назад</button>

        </div>

        <!-- Кнопка закрытия модального окна -->

        <button class="close-modal-btn" @click="emit('close')">Закрыть</button>

      </div>

    </div>

  </template>

  <script setup>

  import { ref, onMounted } from 'vue'

  import api from '@/services/api'  // импорт API для запросов

  const emit = defineEmits(['close']) // событие для закрытия модалки

  const tab = ref('list')             // активная вкладка (список или профиль)

  const users = ref([])               // массив пользователей

  const selectedUser = ref(null)      // выбранный пользователь для редактирования

  const loading = ref(true)           // состояние загрузки

  // Функция загрузки пользователей с сервера

  const fetchUsers = async () => {

    loading.value = **true**

    try {

      const response = await api.get('/user') // запрос списка пользователей

      users.value = response.data             // сохраняем в reactive переменную

    } catch (error) {

      alert('Ошибка загрузки пользователей')  // ошибка запроса

      console.error(error)

    } finally {

      loading.value = **false**                    // отключаем индикатор загрузки

    }

  }

  // При выборе пользователя копируем объект и переключаемся на вкладку профиля

  const selectUser = (user) => {

    selectedUser.value = { ...user } // создаём копию объекта для редактирования

    tab.value = 'detail'             // переключение вкладки

  }

  // Отправка обновлённых данных пользователя на сервер

  const updateUser = async () => {

    try {

      // Формируем данные для отправки (без роли)

      const payload = {

        name: selectedUser.value.name,

        email: selectedUser.value.email,

        telephone: selectedUser.value.telephone,

        sex: selectedUser.value.sex

      }

      // Отправка PATCH-запроса на обновление пользователя

      await api.patch(`/user/${selectedUser.value.id}`, payload)

      alert('Пользователь обновлён')

      await fetchUsers() // обновляем список после изменения

      tab.value = 'list' // возвращаемся к списку

    } catch (error) {

      // Если ошибка валидации, показываем её пользователю

      if (error.response?.status === 422) {

        const messages = Object.values(error.response.data.errors).flat().join('\n')

        alert('Ошибка:\n' + messages)

      } else {

        alert('Ошибка обновления')

        console.error(error)

      }

    }

  }

  // При монтировании компонента загружаем список пользователей

  onMounted(() => {

    fetchUsers()

  })

  </script>

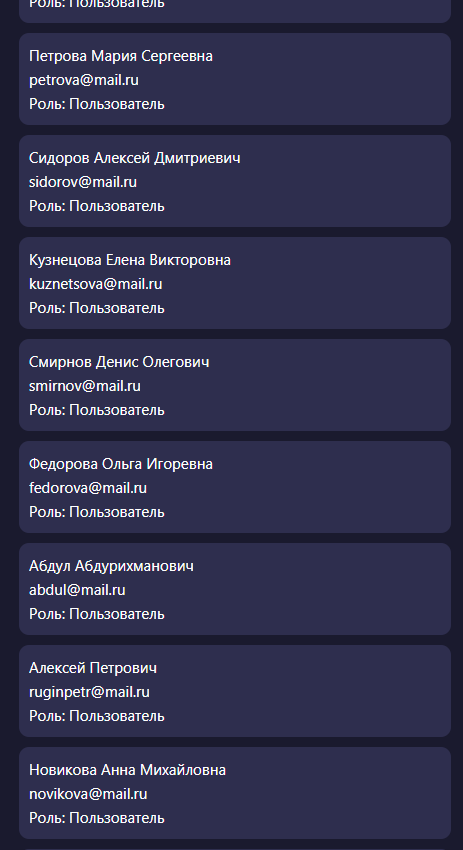
****

Рисунок 26 – Компонент AllUsersModal

**Компонент CreateCategoryModal**

Компонент CreateCategoryModal представляет собой модальное окно, предназначенное для создания новой категории товаров в административной панели.

1. Форма ввода

Содержит единственное обязательное поле — название категории. Значение поля связывается с реактивной переменной form.name.

1. Обработка создания

При отправке формы выполняется POST-запрос к маршруту /category. В процессе запроса отображается индикатор загрузки. После успешного создания появляется уведомление, поле очищается, и окно автоматически закрывается.

Компонент обеспечивает простой и удобный интерфейс для быстрого добавления новых категорий. Визуальное представление приведено на рисунке 27.

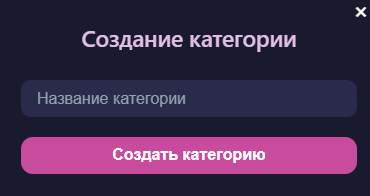


Рисунок 27 – Компонент CreateCategoryModal

**Компонент CreateManagerModal**

Компонент CreateManagerModal реализует интерфейс управления менеджерами в административной панели: просмотр списка, создание и удаление менеджеров.

1. Навигация по вкладкам

В верхней части располагаются две вкладки:

* Список — отображает список всех менеджеров с их именами, email и телефоном.
* Создать — форма для добавления нового менеджера.

1. Просмотр и удаление менеджеров

Во вкладке списка происходит загрузка менеджеров с сервера. Каждый менеджер отображается с кнопкой удаления. При удалении запрашивается подтверждение, после чего менеджер удаляется через API, и список обновляется.

1. Создание менеджера

Форма создания содержит поля: имя, email, пароль, телефон и пол. При отправке формы отправляется POST-запрос на сервер. Ошибки валидации и другие ошибки отображаются пользователю. После успешного создания форма сбрасывается, а пользователь возвращается на вкладку списка с обновлёнными данными.

1. Обработка ошибок

Ошибки загрузки списка, создания и удаления обрабатываются и отображаются через сообщения или alert.

1. Закрытие модального окна

Окно можно закрыть кнопкой или кликом по затемнённой области.

Компонент обеспечивает полный цикл управления менеджерами — просмотр, создание и удаление — в одном удобном интерфейсе. Визуальное представление компонента показано на рисунке 28, 29, а исходный код — в листинге 51.

**Листинг 51. Компонент CreateManagerModal.vue**

<template>

  <div class="modal-overlay" @click.self="emit('close')">

    <div class="modal-content">

      <h2 class="modal-title">Менеджеры</h2>

      <!-- Навигация: переключение вкладок -->

      <div class="tabs">

        <button :class="{active: tab === 'list'}" @click="tab = 'list'">Список</button>

        <button :class="{active: tab === 'create'}" @click="tab = 'create'">Создать</button>

      </div>

      <!-- Ошибка при загрузке списка менеджеров -->

      <div v-if="errorLoading" class="error-message">*{{ errorLoading }}*</div>

      <!-- Вкладка: список менеджеров -->

      <div v-if="tab === 'list'">

        <div v-if="loading">Загрузка...</div>

        <ul v-else class="manager-list">

          <li v-for="manager in managers" :key="manager.id" class="manager-item">

            <div>

              <strong>*{{ manager.name }}*</strong><br />

*{{ manager.email }}*<br />

*{{ manager.telephone }}*

            </div>

            <button class="delete-btn" @click="deleteManager(manager.id)">Удалить</button>

          </li>

        </ul>

      </div>

      <!-- Вкладка: форма создания менеджера -->

      <div v-if="tab === 'create'">

        <input v-model="name" type="text" placeholder="Имя" class="modal-input" />

        <input v-model="email" type="email" placeholder="Email" class="modal-input" />

        <input v-model="password" type="password" placeholder="Пароль" class="modal-input" />

        <input v-model="telephone" type="text" placeholder="Телефон" class="modal-input" />

        <select v-model="sex" class="modal-input">

          <option disabled value="">Выберите пол</option>

          <option value="0">Мужской</option>

          <option value="1">Женский</option>

        </select>

        <!-- Вывод ошибок валидации или API -->

        <div v-if="formError" class="error-message">*{{ formError }}*</div>

        <button class="submit-btn" @click="handleSubmit">Создать</button>

        <button class="cancel-btn" @click="tab = 'list'">Отмена</button>

      </div>

      <!-- Кнопка закрытия модального окна -->

      <button class="close-modal-btn" @click="emit('close')">Закрыть</button>

    </div>

  </div>

</template>

<script setup>

import { ref, onMounted } from 'vue'

import api from '@/services/api'

const emit = defineEmits(['close'])

const tab = ref('list') // Активная вкладка: 'list' или 'create'

const name = ref('')

const email = ref('')

const password = ref('')

const telephone = ref('')

const sex = ref('')

const managers = ref([])

const loading = ref(true)

// Переменные для ошибок

const errorLoading = ref('') // Ошибка загрузки списка

const formError = ref('')     // Ошибка формы создания

// Функция загрузки списка менеджеров с обработкой ошибок

const fetchManagers = async () => {

  loading.value = **true**

  errorLoading.value = ''

  try {

    const response = await api.get('/manager')

    managers.value = response.data

  } catch (e) {

    // Выводим ошибку загрузки

    errorLoading.value = e.response?.data?.message || 'Ошибка загрузки менеджеров'

    console.error('Ошибка загрузки менеджеров:', e)

  } finally {

    loading.value = **false**

  }

}

// Функция создания менеджера с обработкой ошибок

const handleSubmit = async () => {

  formError.value = '' // Сбрасываем ошибки при новом запросе

  try {

    await api.post('/manager', {

      name: name.value,

      email: email.value,

      password: password.value,

      telephone: telephone.value,

      sex: sex.value

    })

    alert('Менеджер создан')

    // Сброс формы после успешного создания

    name.value = ''

    email.value = ''

    password.value = ''

    telephone.value = ''

    sex.value = ''

    tab.value = 'list' // Переход к списку

    await fetchManagers() // Обновление списка

  } catch (error) {

    // Обработка ошибок валидации (422)

    if (error.response?.status === 422) {

      const messages = Object.values(error.response.data.errors).flat().join('\n')

      formError.value = messages

    }

    // Ошибки с кодом 403 или 400 — выводим сообщение из API

    else if ([400, 403].includes(error.response?.status)) {

      formError.value = error.response.data.message || 'Ошибка при выполнении действия'

    } else {

      formError.value = 'Ошибка при создании менеджера'

    }

    console.error('Ошибка создания менеджера:', error)

  }

}

// Функция удаления менеджера с подтверждением и обработкой ошибок

const deleteManager = async (id) => {

  if (!confirm('Удалить менеджера?')) return

  try {

    await api.delete(`/manager/${id}`)

    alert('Менеджер удален')

    await fetchManagers() // Обновление списка после удаления

  } catch (error) {

    // Ошибка удаления — выводим alert и логируем

    alert(error.response?.data?.message || 'Ошибка удаления менеджера')

    console.error('Ошибка удаления менеджера:', error)

  }

}

// Загрузка списка менеджеров при монтировании компонента

onMounted(fetchManagers)

</script>

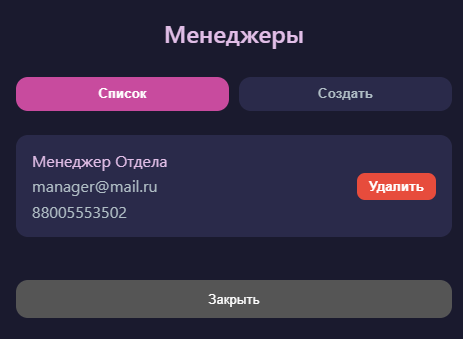
****

Рисунок 28 – Компонент CreateManagerModal

****

Рисунок 29 – Компонент CreateManagerModal (2)

**Компонент CreatePointsModal**

Компонент CreatePointsModalреализует управление пунктами выдачи товаров — просмотр, создание, редактирование и удаление через модальное окно с вкладками.

1. Просмотр списка пунктов

Во вкладке «Список» отображается таблица или список с пунктами выдачи, показывающими город, улицу и номер дома. Для каждого пункта доступны кнопки «Редактировать» и «Удалить». При загрузке данных отображается индикатор загрузки, а в случае ошибки — сообщение об ошибке.

1. Создание нового пункта выдачи

Вкладка «Создать» содержит форму с полями «Город», «Улица» и «Дом». Все поля обязательны. При заполнении и отправке формы вызывается метод handleCreate, который выполняет валидацию и отправляет POST-запрос на API. В случае ошибок отображаются сообщения, полученные от сервера.

1. Редактирование существующего пункта

При нажатии «Редактировать» во вкладке «Список» открывается вкладка «Редактировать», где поля формы заполняются данными выбранного пункта. Метод handleEdit отправляет PATCH-запрос на обновление данных. В случае ошибок также отображаются сообщения.

1. Удаление пункта выдачи

Кнопка «Удалить» вызывает подтверждение удаления. При согласии отправляется DELETE-запрос на сервер. В случае ошибок удаления, например из-за связанных заказов, показывается соответствующее предупреждение.

1. Управление состоянием интерфейса

Компонент реализует переключение между вкладками, сброс форм и ошибок при смене вкладки, а также загрузку актуальных данных при монтировании компонента и после создания, редактирования или удаления пунктов.

Визуальное представление компонента показано на рисунке 30, 31. Полный исходный код компонента приведён в листинге 52.

**Листинг 52. Компонент CreatePointsModal.vue**

<template>

  <div class="modal-overlay" @click.self="emit('close')">

    <div class="modal-content">

      <h2 class="modal-title">Пункты выдачи</h2>

      <!-- Навигация вкладок: список / создание / редактирование -->

      <div class="tabs">

        <button :class="{active: tab === 'list'}" @click="switchTab('list')">Список</button>

        <button :class="{active: tab === 'create'}" @click="switchTab('create')">Создать</button>

        <button v-if="tab === 'edit'" :class="{active: tab === 'edit'}" disabled>Редактировать</button>

      </div>

      <!-- Список пунктов выдачи -->

      <div v-if="tab === 'list'">

        <div v-if="loading">Загрузка...</div>

        <ul v-else class="points-list">

          <li v-for="point in points" :key="point.id" class="point-item">

            <div>

              <strong>*{{ point.city }}*, *{{ point.street }}*, д. *{{ point.house }}*</strong>

            </div>

            <div>

              <button class="edit-btn" @click="startEdit(point)">Редактировать</button>

              <button class="delete-btn" @click="deletePoint(point.id)">Удалить</button>

            </div>

          </li>

        </ul>

      </div>

      <!-- Форма создания -->

      <div v-if="tab === 'create'">

        <input v-model="city" type="text" placeholder="Город" class="modal-input" />

        <input v-model="street" type="text" placeholder="Улица" class="modal-input" />

        <input v-model="house" type="text" placeholder="Дом" class="modal-input" />

        <!-- Отображение ошибок -->

        <div v-if="errorMessageCreate" class="error-message">*{{ errorMessageCreate }}*</div>

        <button class="submit-btn" @click="handleCreate">Создать</button>

        <button class="cancel-btn" @click="switchTab('list')">Отмена</button>

      </div>

      <!-- Форма редактирования -->

      <div v-if="tab === 'edit'">

        <input v-model="city" type="text" placeholder="Город" class="modal-input" />

        <input v-model="street" type="text" placeholder="Улица" class="modal-input" />

        <input v-model="house" type="text" placeholder="Дом" class="modal-input" />

        <!-- Отображение ошибок -->

        <div v-if="errorMessageEdit" class="error-message">*{{ errorMessageEdit }}*</div>

        <button class="submit-btn" @click="handleEdit">Сохранить</button>

        <button class="cancel-btn" @click="switchTab('list')">Отмена</button>

      </div>

      <!-- Кнопка закрыть -->

      <button class="close-modal-btn" @click="emit('close')">Закрыть</button>

    </div>

  </div>

</template>

<script setup>

import { ref, onMounted } from 'vue'

import api from '@/services/api'

const emit = defineEmits(['close'])

const tab = ref('list')

const points = ref([])

const loading = ref(false)

// Поля формы (общие для создания и редактирования)

const city = ref('')

const street = ref('')

const house = ref('')

const editingId = ref(null) // ID редактируемого пункта

// Переменные для сообщений об ошибках (для двух форм отдельно)

const errorMessageCreate = ref('')

const errorMessageEdit = ref('')

// Получение списка пунктов выдачи с сервера

const fetchPoints = async () => {

  loading.value = **true**

  try {

    const response = await api.get('/point')

    points.value = response.data

  } catch (error) {

    errorMessageCreate.value = 'Ошибка загрузки пунктов выдачи'

    errorMessageEdit.value = 'Ошибка загрузки пунктов выдачи'

    console.error(error)

  } finally {

    loading.value = **false**

  }

}

// Смена вкладки и очистка ошибок и формы при смене

const switchTab = (newTab) => {

  tab.value = newTab

  clearForm()

  editingId.value = **null**

  errorMessageCreate.value = ''

  errorMessageEdit.value = ''

}

// Очистка полей формы

const clearForm = () => {

  city.value = ''

  street.value = ''

  house.value = ''

}

// Создание нового пункта выдачи с обработкой ошибок из API

const handleCreate = async () => {

  // Проверка заполнения всех полей

  if (!city.value || !street.value || !house.value) {

    errorMessageCreate.value = 'Заполните все поля'

    return

  }

  errorMessageCreate.value = '' // Сброс ошибки перед запросом

  try {

    await api.post('/point', {

      city: city.value,

      street: street.value,

      house: house.value

    })

    // Успешное создание — переход к списку и обновление данных

    switchTab('list')

    await fetchPoints()

  } catch (error) {

    // Обработка ошибок, приходящих из Laravel API

    if (error.response?.status === 422) {

      // Валидационные ошибки

      const messages = Object.values(error.response.data.errors).flat().join('\n')

      errorMessageCreate.value = messages

    } else if (error.response?.status === 403) {

      errorMessageCreate.value = 'Нет прав для создания пункта выдачи'

    } else {

      errorMessageCreate.value = 'Ошибка при создании пункта выдачи'

      console.error(error)

    }

  }

}

// Инициализация формы редактирования пунктом, заполнение полей

const startEdit = (point) => {

  editingId.value = point.id

  city.value = point.city

  street.value = point.street

  house.value = point.house

  errorMessageEdit.value = ''

  tab.value = 'edit'

}

// Сохранение изменений редактируемого пункта с обработкой ошибок

const handleEdit = async () => {

  if (!city.value || !street.value || !house.value) {

    errorMessageEdit.value = 'Заполните все поля'

    return

  }

  errorMessageEdit.value = ''

  try {

    await api.patch(`/point/${editingId.value}`, {

      city: city.value,

      street: street.value,

      house: house.value

    })

    // После успешного обновления — возврат к списку и обновление данных

    switchTab('list')

    await fetchPoints()

  } catch (error) {

    if (error.response?.status === 422) {

      const messages = Object.values(error.response.data.errors).flat().join('\n')

      errorMessageEdit.value = messages

    } else if (error.response?.status === 403) {

      errorMessageEdit.value = 'Нет прав для редактирования пункта выдачи'

    } else if (error.response?.status === 404) {

      errorMessageEdit.value = 'Пункт выдачи не найден'

    } else {

      errorMessageEdit.value = 'Ошибка при обновлении пункта выдачи'

      console.error(error)

    }

// Удаление пункта с подтверждением и обработкой ошибок

const deletePoint = async (id) => {

  if (!confirm('Удалить пункт выдачи?')) return

  try {

    await api.delete(`/point/${id}`)

    await fetchPoints()

  } catch (error) {

    if (error.response?.status === 409) {

      alert('Нельзя удалить пункт выдачи: есть связанные заказы')

    } else if (error.response?.status === 403) {

      alert('Нет прав на удаление')

    } else if (error.response?.status === 404) {

      alert('Пункт выдачи не найден')

    } else {

      alert('Ошибка при удалении пункта')

      console.error(error)

    }

  }

}

onMounted(fetchPoints)

</script>

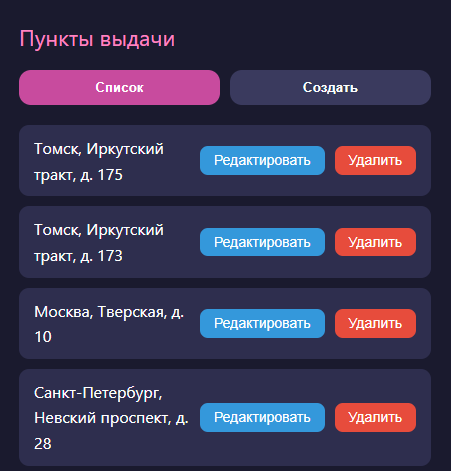
****

Рисунок 30 - Компонент CreatePointsModal

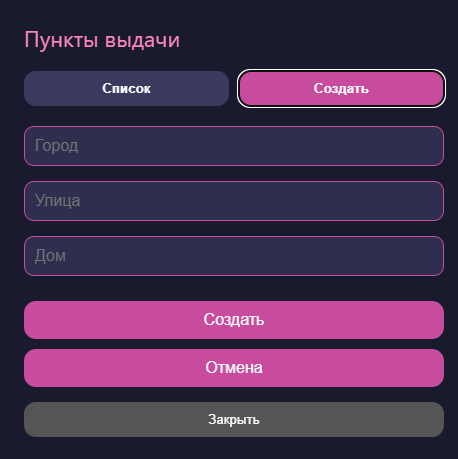
****

Рисунок 32 - Компонент CreatePointsModal (2)

**Компонент UserOrdersModal**

Компонент UserOrdersModal реализует отображение истории заказов всех пользователей с возможностью просмотра подробностей и изменения статуса (для администраторов/менеджеров). Визуально компонент представлен в виде модального окна с вкладками.

1. Просмотр списка заказов

После открытия модального окна по умолчанию активна вкладка «Список заказов». Здесь отображаются все заказы, полученные с API /order. Для каждого заказа выводится:

* Номер заказа
* Статус
* Дата оформления
* Адрес доставки
* Общая сумма

Если список пуст — отображается сообщение «Нет заказов». При клике на заказ загружается подробная информация (см. следующий пункт).

1. Детали заказа

Во вкладке «Детали заказа» отображаются:

* Информация о заказе: номер, дата, адрес и общая сумма;
* Селект для изменения статуса;
* Список заказанных товаров с указанием:
* Названия, количества и цены,
* Цвета и размера (если указаны),
* Фото и итоговой суммы за позицию.
* Кнопка «Назад к списку» позволяет вернуться к предыдущей вкладке.

1. Управление статусом заказа

Для изменения статуса заказа предусмотрен выпадающий список со статусами, полученными с /statuses. При выборе нового значения и потверждении выполняется PATCH-запрос на /order/{id}. После обновления:

* Статус сохраняется на сервере,
* Повторно загружаются детали и список заказов,
* Пользователю отображается уведомление об успешном обновлении.

1. Работа с API и обработка состояний

Компонент использует следующие методы:

* fetchOrders() — загрузка списка заказов;
* loadOrderDetails(id) — загрузка детальной информации по конкретному заказу;
* fetchStatuses() — загрузка возможных статусов заказа;
* updateOrderStatus() — отправка обновлённого статуса.

Во всех запросах реализована обработка ошибок и отображение соответствующих сообщений.

1. Интерфейс и пользовательский опыт

Компонент реализован в виде модального окна (modal-overlay), закрывается при клике вне окна или кнопкой «Закрыть».

Визуально использованы вкладки (tabs) для переключения между списком заказов и детальной информацией.

Загружаемые данные форматируются перед отображением (formatDate, formatAddress), чтобы улучшить читаемость.

Таким образом, компонент UserOrders обеспечивает интуитивно понятный интерфейс для просмотра и управления заказами пользователей, включая подробный состав заказа и возможность редактирования его статуса.

Визуальное отображение компонента представлено на рисунке 33, 34. Исходный код компонента приведён в листинге 53.

**Листинг 53. UserOrdersModal.vue**

<template>

  <div class="modal-overlay" @click.self="emit('close')">

    <div class="modal-content">

      <h2 class="modal-title">Мои заказы</h2>

      <!-- Вкладки -->

      <div class="tabs">

        <button :class="{ active: tab === 'list' }" @click="tab = 'list'">

          Список заказов

        </button>

        <button v-if="selectedOrder" :class="{ active: tab === 'details' }" @click="tab = 'details'">

          Детали заказа

        </button>

      </div>

      <!-- Список заказов -->

      <div v-if="tab === 'list'">

        <div v-if="loading">Загрузка...</div>

        <div v-else-if="!orders.length">Нет заказов</div>

        <ul v-else class="order-list">

          <li v-for="order in orders" :key="order.id" class="order-item" @click="loadOrderDetails(order.id)">

            <strong>Заказ №*{{ order.id }}*</strong><br />

            Статус: *{{ order.status }}*<br />

            Дата: *{{ formatDate(order.order\_date) }}*<br />

            Адрес: *{{ formatAddress(order.address) }}*<br />

            Сумма: *{{ order.total\_cost }}* ₽

          </li>

        </ul>

      </div>

      <!-- Детали заказа -->

      <div v-if="tab === 'details' && selectedOrder" class="order-details">

        <h3>Заказ №*{{ selectedOrder.id }}*</h3>

        <p>Дата: *{{ formatDate(selectedOrder.order\_date) }}*</p>

        <p>Адрес: *{{ formatAddress(selectedOrder.address) }}*</p>

        <p>Итого: *{{ selectedOrder.total\_cost }}* ₽</p>

        <!-- Селект для смены статуса -->

        <label>Статус:</label>

        <select v-model="selectedStatusId" @change="updateOrderStatus">

          <option v-for="status in statuses" :key="status.id" :value="status.id">

*{{ status.name }}*

          </option>

        </select>

        <h4>Товары:</h4>

        <ul>

          <li v-for="item in selectedOrder.items" :key="item.product\_id" class="item-block">

            <img :src="item.photo" alt="Фото товара" />

            <div>

              <strong>*{{ item.name }}*</strong><br />

*{{ item.quantity }} шт. по {{ item.price }}* ₽<br />

              Цвет: *{{ item.color || '—' }}*, Размер: *{{ item.size || '—' }}*<br />

              Сумма: *{{ item.total }}* ₽

            </div>

          </li>

        </ul>

        <button @click="tab = 'list'" class="back-btn">Назад к списку</button>

      </div>

      <button class="close-modal-btn" @click="emit('close')">Закрыть</button>

    </div>

  </div>

</template>

<script setup>

import { ref, onMounted } from 'vue'

import api from '@/services/api'

const emit = defineEmits(['close'])

const tab = ref('list')

const orders = ref([])

const selectedOrder = ref(null)

const loading = ref(false)

const statuses = ref([])

const selectedStatusId = ref(null)

// Получение списка заказов

async function fetchOrders() {

  loading.value = **true**

  try {

    const res = await api.get('/order')

    orders.value = Array.isArray(res.data) ? res.data : res.data.data || []

  } catch (e) {

    alert('Ошибка загрузки заказов')

    console.error(e)

  } finally {

    loading.value = **false**

  }

}

// Загрузка деталей заказа

async function loadOrderDetails(id) {

  loading.value = **true**

  try {

    const res = await api.get(`/order/${id}`)

    selectedOrder.value = res.data

    selectedStatusId.value = res.data.status\_id // выбранный статус

    tab.value = 'details'

  } catch (e) {

    alert('Ошибка при загрузке деталей')

    console.error(e)

  } finally {

    loading.value = **false**

  }

}

// Загрузка списка возможных статусов

async function fetchStatuses() {

  try {

    const res = await api.get('/statuses') // <--- Исправлено: правильный путь

    statuses.value = Array.isArray(res.data) ? res.data : res.data.data || []

  } catch (e) {

    console.error('Ошибка загрузки статусов', e)

  }

}

// Обновление статуса заказа

async function updateOrderStatus() {

  try {

    await api.patch(`/order/${selectedOrder.value.id}`, {

      status\_id: selectedStatusId.value,

    })

    // Повторная загрузка данных для актуализации

    await loadOrderDetails(selectedOrder.value.id)

    await fetchOrders()

    alert('Статус обновлён')

  } catch (e) {

    console.error('Ошибка обновления статуса', e)

    alert('Не удалось обновить статус')

  }

}

// Формат даты

function formatDate(dateStr) {

  if (!dateStr) return ''

  const d = new Date(dateStr)

  return d.toLocaleDateString('ru-RU', {

    day: '2-digit',

    month: '2-digit',

    year: 'numeric',

  }) + ' ' + d.toLocaleTimeString('ru-RU', {

    hour: '2-digit',

    minute: '2-digit',

  })

}

// Формат адреса

function formatAddress(addr) {

  if (!addr) return '—'

  if (typeof addr === 'string') return addr

  return [addr.city, addr.street, addr.house].filter(Boolean).join(', ')

}

// Первичная загрузка

onMounted(() => {

  fetchOrders()

  fetchStatuses()

})

</script>

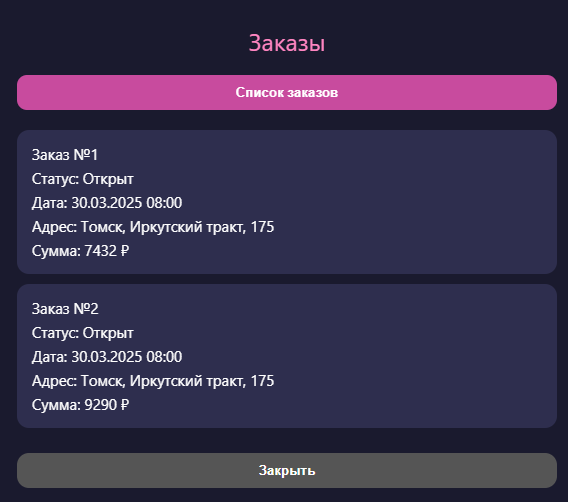
****

Рисунок 33 – Компонент UserOrdersModal

****

Рисунок 34 - Компонент UserOrdersModal (2)

**Компонент CreateProductModal**

Компонент CreateProductModal реализует форму создания нового товара. Он позволяет пользователю ввести все необходимые параметры товара, загрузить фотографию, выбрать или создать цвета и размеры, а также отправить информацию на сервер. Компонент построен на основе FormData и активно использует реактивные переменные и списки.

1. Основные поля товара

Форма включает поля:

* Название товара — текстовое поле;
* Описание — многострочное текстовое поле;
* Пол — выбор между женским и мужским;
* Цена, Категория, Страна — числовое и выпадающие поля;
* Фото — загрузка изображения, сохраняется как файл в FormData.

Данные для полей Категория, Страна, Цвета, Размеры подгружаются при монтировании компонента с сервера.

1. Добавление цветов и размеров

Товар может иметь несколько цветовых вариантов. Каждый цвет:

* Либо выбирается из существующих (color\_id),
* Либо создаётся новый (new\_color\_name, new\_color\_hex).

Для каждого цвета можно задать один или несколько размеров. Аналогично:

* Либо выбирается существующий (size\_id),
* Либо добавляется новый (new\_size\_name).

Каждый размер требует указания количества (quantity), начиная от 1.

Пользователь может добавлять и удалять как цвета, так и размеры внутри них.

1. Управление формой

При изменении файла изображения вызывается метод handleFileChange, сохраняющий файл в form.photo.

Добавление/удаление цвета и размеров осуществляется с помощью addColor, removeColor, addSize, removeSize. Все новые элементы создаются через вспомогательную функцию createSize().

1. Отправка данных

После заполнения всех полей, при нажатии кнопки «Создать товар», вызывается метод handleSubmit, который:

* Формирует FormData, включая все поля, фото, цвета и размеры;
* Отправляет POST-запрос на /product с Content-Type: multipart/form-data;
* Обрабатывает ошибки валидации и отображает их пользователю.

Ошибки разбиваются с помощью утилиты flattenErrors, чтобы корректно отобразить вложенные ошибки, включая ошибки массивов colors и sizes.

Компонент CreateProductModal охватывает весь процесс создания нового товара, включая расширенную поддержку вложенных данных (цвета/размеры) и валидации. Это делает его универсальным инструментом для менеджера или администратора магазина.

Визуальное отображение компонента представлено на рисунке 35, 36 а его полный исходный код приведён в листинге 54.

**Листинг 54. Компонент CreateProductModal.vue**

<template>

  <!-- Клик по оверлею вызывает закрытие модалки -->

  <div class="modal-overlay" @click.self="close">

    <div class="modal-content">

      <h2 class="title">Создание товара</h2>

      <div class="modal-scrollable-content">

        <form @submit.prevent="handleSubmit" enctype="multipart/form-data">

          <!-- Название товара -->

          <label>Название товара

            <input v-model="form.name" type="text" placeholder="Название товара" required />

          </label>

          <!-- Описание -->

          <label>Описание

            <textarea v-model="form.description" placeholder="Описание"></textarea>

          </label>

          <!-- Пол -->

          <label>Пол

            <select v-model.number="form.sex" required>

              <option disabled value="">Выберите пол</option>

              <option :value="0">Женский</option>

              <option :value="1">Мужской</option>

            </select>

          </label>

          <!-- Цена -->

          <label>Цена

            <input v-model.number="form.price" type="number" placeholder="Цена" min="0" required />

          </label>

          <!-- Категория -->

          <label>Категория

            <select v-model="form.category\_id" required>

              <option disabled value="">Выберите категорию</option>

              <option v-for="cat in categories" :key="cat.id" :value="cat.id">*{{ cat.name }}*</option>

            </select>

          </label>

          <!-- Страна -->

          <label>Страна

            <select v-model="form.country\_id" required>

              <option disabled value="">Выберите страну</option>

              <option v-for="country in countries" :key="country.id" :value="country.id">*{{ country.name }}*</option>

            </select>

          </label>

          <!-- Фото -->

          <label>Фото

            <input type="file" @change="handleFileChange" />

          </label>

          <!-- Цвета и размеры -->

          <div v-for="(color, cIndex) in form.colors" :key="cIndex" class="color-size-row">

            <div class="button-group">

              <button

                type="button"

                :class="{ active: !color.isNewColor }"

                @click="color.isNewColor = false"

              >Выбрать цвет</button>

              <button

                type="button"

                :class="{ active: color.isNewColor }"

                @click="color.isNewColor = true"

              >Новый цвет</button>

            </div>

            <div v-if="!color.isNewColor" class="select-wrapper">

              <select v-model="color.color\_id" required>

                <option disabled value="">Выберите цвет</option>

                <option v-for="c in availableColors" :key="c.id" :value="c.id">*{{ c.name }} ({{ c.hex }}*)</option>

              </select>

            </div>

            <div v-if="color.isNewColor" class="new-color-inputs">

              <input v-model="color.new\_color\_name" type="text" placeholder="Название цвета" required />

              <input

                v-model="color.new\_color\_hex"

                type="text"

                placeholder="#HEX код"

                required

                pattern="^#[0-9A-Fa-f]{6}$"

              />

            </div>

            <div

              v-for="(size, sIndex) in color.sizes"

              :key="sIndex"

              class="color-size-row"

              style="margin-top: 10px;"

            >

              <div class="button-group" style="flex-wrap: nowrap;">

                <button

                  type="button"

                  :class="{ active: !size.isNewSize }"

                  @click="size.isNewSize = false"

                  class="small-btn"

                >Выбрать размер</button>

                <button

                  type="button"

                  :class="{ active: size.isNewSize }"

                  @click="size.isNewSize = true"

                  class="small-btn"

                >Новый размер</button>

              </div>

              <div v-if="!size.isNewSize" class="select-wrapper" style="flex: 1;">

                <select v-model="size.size\_id" required>

                  <option disabled value="">Выберите размер</option>

                  <option v-for="s in availableSizes" :key="s.id" :value="s.id">*{{ s.name }}*</option>

                </select>

              </div>

              <div v-if="size.isNewSize" class="new-color-inputs" style="flex: 1;">

                <input v-model="size.new\_size\_name" type="text" placeholder="Новый размер" required />

              </div>

              <input

                type="number"

                v-model.number="size.quantity"

                min="1"

                placeholder="Кол-во"

                class="quantity-input"

                required

                style="width: 90px;"

              />

              <button

                @click.prevent="removeSize(cIndex, sIndex)"

                class="remove-btn"

                title="Удалить размер"

              >×</button>

            </div>

            <div

              class="button-group"

              style="margin-top: 8px; justify-content: flex-start; gap: 10px;"

            >

              <button @click.prevent="addSize(cIndex)" type="button">+ Размер</button>

              <button

                @click.prevent="removeColor(cIndex)"

                type="button"

                class="remove-btn"

              >Удалить цвет</button>

            </div>

          </div>

          <button @click.prevent="addColor" class="add-color-size-btn" type="button">+ Добавить цвет</button>

          <!-- Кнопки формы -->

          <div class="buttons" style="margin-top: 20px; display: flex; gap: 10px;">

            <button type="submit">Создать товар</button>

            <button type="button" @click="close" class="cancel-btn">Отмена</button>

          </div>

          <!-- Ошибки -->

          <div v-if="errors.general" class="error-message">*{{ errors.general }}*</div>

          <div v-for="(err, key) in errors" :key="key" class="error-message">

*{{ key }}*: {{ Array.isArray(err) ? err.join(', ') : err }}

          </div>

        </form>

      </div>

    </div>

  </div>

</template>

<script setup>

import { reactive, ref, onMounted } from 'vue'

import api from '@/services/api'

const emit = defineEmits(['close'])

const form = reactive({

  name: '',

  description: '',

  sex: **null**,

  price: **null**,

  category\_id: '',

  country\_id: '',

  photo: **null**,

  colors: []

})

const errors = reactive({})

const categories = ref([])

const countries = ref([])

const availableColors = ref([])

const availableSizes = ref([])

onMounted(async () => {

  try {

    const [catRes, countryRes, colorsRes, sizesRes] = await Promise.all([

      api.get('/category'),

      api.get('/country'),

      api.get('/colors'),

      api.get('/sizes')

    ])

    categories.value = catRes.data

    countries.value = countryRes.data

    availableColors.value = colorsRes.data

    availableSizes.value = sizesRes.data

    addColor()

  } catch {

    alert('Ошибка загрузки данных')

  }

})

function handleFileChange(e) {

  form.photo = e.target.files[0]

}

function addColor() {

  form.colors.push({

    isNewColor: **false**,

    color\_id: '',

    new\_color\_name: '',

    new\_color\_hex: '',

    sizes: [createSize()]

  })

}

function removeColor(index) {

  form.colors.splice(index, 1)

}

function addSize(colorIndex) {

  form.colors[colorIndex].sizes.push(createSize())

}

function removeSize(colorIndex, sizeIndex) {

  form.colors[colorIndex].sizes.splice(sizeIndex, 1)

}

function createSize() {

  return {

    isNewSize: **false**,

    size\_id: '',

    new\_size\_name: '',

    quantity: 1

  }

}

function flattenErrors(errorObj, prefix = '', target = {}) {

  for (const key in errorObj) {

    const value = errorObj[key]

    const newPrefix = prefix ? `${prefix}.${key}` : key

    if (Array.isArray(value) && typeof value[0] === 'string') {

      target[newPrefix] = value

    } else if (typeof value === 'object' && value !== null) {

      flattenErrors(value, newPrefix, target)

    }

  }

  return target

}

async function handleSubmit() {

  for (const key in errors) delete errors[key]

  const formData = new FormData()

  formData.append('name', form.name)

  formData.append('description', form.description)

  formData.append('sex', form.sex)

  formData.append('price', form.price)

  formData.append('category\_id', form.category\_id)

  formData.append('country\_id', form.country\_id)

  if (form.photo) formData.append('photo', form.photo)

  form.colors.forEach((color, cIndex) => {

    if (color.isNewColor) {

      formData.append(`colors[${cIndex}][new\_color\_name]`, color.new\_color\_name)

      formData.append(`colors[${cIndex}][new\_color\_hex]`, color.new\_color\_hex)

    } else {

      formData.append(`colors[${cIndex}][color\_id]`, color.color\_id)

    }

    color.sizes.forEach((size, sIndex) => {

      if (size.isNewSize) {

        formData.append(`colors[${cIndex}][sizes][${sIndex}][new\_size\_name]`, size.new\_size\_name)

      } else {

        formData.append(`colors[${cIndex}][sizes][${sIndex}][size\_id]`, size.size\_id)

      }

      formData.append(`colors[${cIndex}][sizes][${sIndex}][quantity]`, size.quantity)

    })

  })

  try {

    await api.post('/product', formData, {

      headers: {'Content-Type': 'multipart/form-data'}

    })

    alert('Товар успешно создан')

    close()

  } catch (err) {

    if (err.response?.status === 422) {

      Object.assign(errors, flattenErrors(err.response.data.errors))

    } else {

      errors.general = 'Ошибка при создании товара'

    }

  }

}

function close() {

  emit('close')

}

</script>

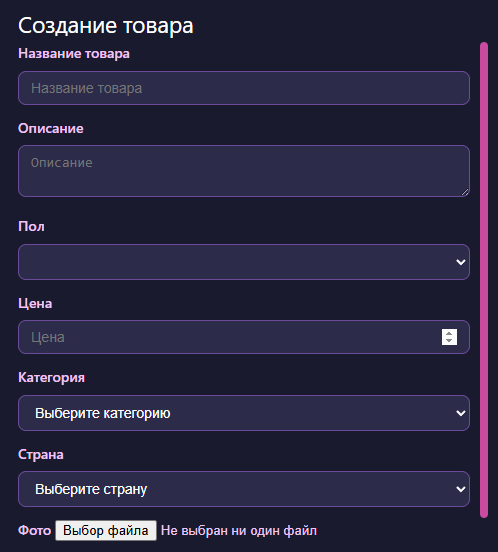
****

Рисунок 35 - Компонент CreateProductModal

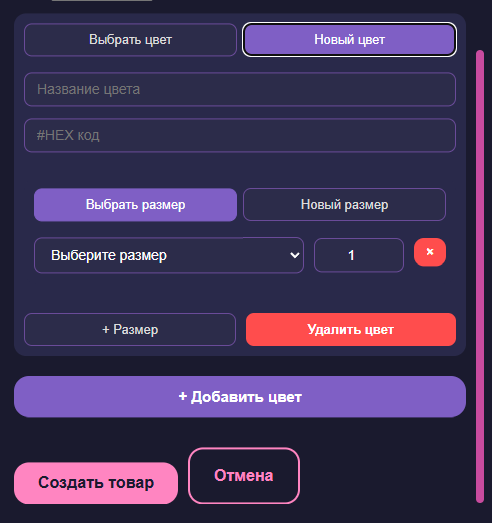
****

Рисунок 36 - Компонент CreateProductModal (2)

**Компонент EditProductModal**

Компонент EditProductModal предназначен для редактирования информации о товаре. Он отображает модальное окно с возможностью изменения всех параметров товара, включая название, описание, цену, пол, категорию, страну, а также цветовые и размерные модификации с учётом количества.

1. Загрузка и инициализация данных

При открытии компонента происходит:

* Загрузка справочной информации: категории, страны, цвета, размеры;
* Загрузка конкретного товара по productId и заполнение полей формы данными из ответа;
* Инициализация поля изображения, если оно уже существует у товара.

1. Основные поля формы

Форма редактирования включает в себя:

* Текстовые поля для названия и описания;
* Поле выбора пола (sex);
* Числовое поле цены;
* Выпадающие списки для выбора категории и страны;

Блок загрузки изображения, отображающий текущее фото, если оно есть, с возможностью его удаления или замены.

1. Редактирование цветов и размеров

Каждая модификация товара представляет собой сочетание цвета и размера:

* Пользователь может выбрать существующий цвет из справочника или ввести новый (имя и hex-код);
* Аналогично для размеров — выбор существующего или создание нового;

Для каждой пары цвет-размер указывается количество на складе.

Пользователь может добавлять или удалять произвольное количество таких комбинаций.

1. Обработка изображения

Если пользователь выбирает новое изображение, оно:

* Проверяется по типу и размеру;
* Сохраняется в локальную переменную и отображается прогресс загрузки;

Удаление текущего изображения также доступно по кнопке.

1. Сборка и отправка формы

После внесения изменений:

* Все данные преобразуются в объект, включая структуру цветов с размерами;
* При необходимости фото отправляется отдельно;

Выполняется PATCH-запрос на сервер с обновлённой информацией.

Если операция проходит успешно — окно закрывается, событие updated отправляется родителю.

1. Дополнительные функции

* Вычисление общего количества товаров во всех комбинациях цвета и размера;
* Кнопки переключения между выбором существующих и созданием новых цветов/размеров;
* Валидация всех обязательных полей до отправки;

Отображение сообщений об ошибках при загрузке данных или сохранении.

Таким образом, EditProductModal — это полнофункциональное модальное окно, позволяющее гибко и удобно редактировать информацию о товаре. Компонент поддерживает как простое редактирование, так и расширенную логику изменения цветоразмерных комбинаций и управления изображением.

Полный код компонента приведён в листинге 55, а его внешний вид идентичен как в создании товара показан на рисунке 35, 36.

**Листинг 55. Компонент EditProductModal.vue**

<template>

  <div class="modal-overlay" @click.self="close">

    <div class="modal-content">

      <button class="close-button" @click="close">×</button>

      <h2>Редактирование товара</h2>

      <div v-if="loading">Загрузка товара...</div>

      <div v-else-if="error" class="error-message">*{{ error }}*</div>

      <div v-else>

        <div class="modal-scrollable-content">

          <form @submit.prevent="submitForm">

            <!-- Блок для загрузки фото -->

            <div class="photo-section">

              <h3>Фотография товара</h3>

              <div v-if="currentPhoto" class="photo-preview">

                <img :src="currentPhoto" alt="Текущее фото товара" class="current-photo" />

                <button type="button" @click="removePhoto" class="remove-photo-btn">

                  Удалить фото

                </button>

              </div>

              <div v-else class="no-photo">Фото не загружено</div>

              <label class="file-upload-label">

                <input

                  type="file"

                  ref="fileInput"

                  accept="image/jpeg, image/png, image/webp"

                  @change="handleFileChange"

                  style="display: none;"

                />

                <span class="upload-btn">

                  {{ selectedFile ? 'Файл выбран' : 'Выберите новое фото' }}

                </span>

                <span v-if="selectedFile" class="file-name">*{{ selectedFile.name }}*</span>

              </label>

              <div v-if="uploadProgress > 0" class="progress-container">

                <div class="progress-bar" :style="{ width: uploadProgress + '%' }"></div>

                <span class="progress-text">*{{ uploadProgress }}*%</span>

              </div>

            </div>

            <div v-if="error" class="error-message">

*{{ error }}*

              <div v-if="error.details" v-for="(detail, key) in error.details" :key="key">

                {{ detail.join(', ') }}

              </div>

            </div>

            <!-- Основные поля формы -->

            <label>

              Название:

              <input type="text" v-model="form.name" required />

            </label>

            <label>

              Описание:

              <textarea v-model="form.description" rows="3"></textarea>

            </label>

            <label>

              Цена:

              <input type="number" v-model.number="form.price" min="0" required />

            </label>=

            <label>

              Пол (sex):

              <select v-model.number="form.sex">

                <option :value="0">Женский</option>

                <option :value="1">Мужской</option>

              </select>

            </label>

            <label>

              Категория:

              <select v-model.number="form.category\_id" required>

                <option v-for="cat in categories" :key="cat.id" :value="cat.id">

*{{ cat.name }}*

                </option>

              </select>

            </label>

            <label>

              Страна:

              <select v-model.number="form.country\_id" required>

                <option v-for="country in countries" :key="country.id" :value="country.id">

*{{ country.name }}*

                </option>

              </select>

            </label>

            <hr />

            <!-- Цвета и размеры -->

            <div class="colors-header">

              <h3>Цвета и размеры (остатки)</h3>

              <div class="total-quantity">

                Общее количество: <strong>*{{ totalQuantity }}*</strong>

              </div>

            </div>=

            <div

              v-for="(pcs, index) in form.product\_color\_sizes"

              :key="pcs.id || index"

              class="color-size-row"

            >

              <div class="button-group">

                <button

                  type="button"

                  :class="{ 'active': !pcs.isNewColor }"

                  @click="pcs.isNewColor = false"

                >

                  Существующий цвет

                </button>

                <button

                  type="button"

                  :class="{ 'active': pcs.isNewColor }"

                  @click="pcs.isNewColor = true"

                >

                  Новый цвет

                </button>

              </div>

              <div v-if="!pcs.isNewColor" class="select-wrapper">

                <select v-model.number="pcs.color\_id" required>

                  <option value="" disabled>Выберите цвет</option>

                  <option v-for="color in colors" :key="color.id" :value="color.id">

*{{ color.name }}*

                  </option>

                </select>

              </div>

              <div v-else class="new-color-inputs">

                <input

                  type="text"

                  v-model="pcs.new\_color\_name"

                  placeholder="Название цвета"

                  required

                />

                <input

                  type="color"

                  v-model="pcs.new\_color\_hex"

                  title="Выберите цвет"

                  required

                />

              </div>

              <div class="button-group">

                <button

                  type="button"

                  :class="{ 'active': !pcs.isNewSize }"

                  @click="pcs.isNewSize = false"

                >

                  Существующий размер

                </button>

                <button

                  type="button"

                  :class="{ 'active': pcs.isNewSize }"

                  @click="pcs.isNewSize = true"

                >

                  Новый размер

                </button>

              </div>

              <div v-if="!pcs.isNewSize" class="select-wrapper">

                <select v-model.number="pcs.size\_id" required>

                  <option value="" disabled>Выберите размер</option>

                  <option v-for="size in sizes" :key="size.id" :value="size.id">

*{{ size.name }}*

                  </option>

                </select>

              </div>

              <div v-else>

                <input

                  type="text"

                  v-model="pcs.new\_size\_name"

                  placeholder="Название размера"

                  required

                />

              </div>

              <input

                type="number"

                v-model.number="pcs.quantity"

                min="0"

                placeholder="Количество"

                required

                class="quantity-input"

              />

              <button type="button" @click="removeColorSize(index)" class="remove-btn">

                Удалить

              </button>

            </div>

            <button

              type="button"

              class="add-color-size-btn"

              @click="addColorSize"

            >

              Добавить цвет и размер

            </button>

            <hr />

            <button type="submit" :disabled="loadingSubmit || isUploadingPhoto">

              {{ loadingSubmit ? 'Сохранение...' : 'Сохранить изменения' }}

            </button>

          </form>

        </div>

      </div>

    </div>

  </div>

</template>

<script setup>

import { ref, computed, watch, onMounted } from 'vue';

import api from '@/services/api';

import axios from 'axios';

const props = defineProps({

  productId: {

    type: [Number, String],

    required: **true**,

  },

});

const emits = defineEmits(['close', 'updated']);

// Состояние компонента

const loading = ref(false);

const loadingSubmit = ref(false);

const error = ref(null);

const isUploadingPhoto = ref(false);

const uploadProgress = ref(0);

const selectedFile = ref(null);

const fileInput = ref(null);

const currentPhoto = ref(null);

// Справочные данные

const categories = ref([]);

const countries = ref([]);

const colors = ref([]);

const sizes = ref([]);

// Форма товара

const form = ref({

  name: '',

  description: '',

  price: 0,

  sex: 0,

  photo: **null**,

  category\_id: **null**,

  country\_id: **null**,

  product\_color\_sizes: [],

});

// Загрузка справочных данных

async function loadReferenceData() {

  try {

    const [catRes, countryRes, colorRes, sizeRes] = await Promise.all([

      api.get('/category'),

      api.get('/country'),

      api.get('/colors'),

      api.get('/sizes'),

    ]);

    categories.value = catRes.data;

    countries.value = countryRes.data;

    colors.value = colorRes.data;

    sizes.value = sizeRes.data;

  } catch (e) {

    error.value = 'Ошибка загрузки справочных данных';

    console.error(e);

  }

}

// Загрузка данных товара

async function fetchProduct(id) {

  loading.value = true;

  error.value = null;

  try {

    const response = await api.get(`/product/${id}`);

    const product = response.data;

    form.value = {

      name: product.name || '',

      description: product.description || '',

      price: Number(product.price) || 0,

      sex: product.sex || 0,

      photo: product.photo || **null**,

      category\_id: product.category\_id || **null**,

      country\_id: product.country\_id || **null**,

      product\_color\_sizes: (product.product\_color\_sizes || []).map(pcs => ({

        id: pcs.id,

        color\_id: pcs.color\_id,

        size\_id: pcs.size\_id,

        quantity: pcs.quantity,

        isNewColor: **false**,

        new\_color\_name: '',

        new\_color\_hex: '#000000',

        isNewSize: **false**,

        new\_size\_name: '',

      })),

    };

    currentPhoto.value = product.photo;

  } catch (e) {

    error.value = 'Ошибка загрузки товара';

    console.error(e);

  } finally {

    loading.value = false;

  }

}

// Обработчик выбора файла

function handleFileChange(event) {

  const file = event.target.files[0];

  if (!file) return;

  // Проверка типа файла

  const validTypes = ['image/jpeg', 'image/png', 'image/webp'];

  if (!validTypes.includes(file.type)) {

    error.value = 'Допустимы только JPEG, PNG и WebP';

    return;

  }

  // Проверка размера (например, 5MB)

  if (file.size > 5 \* 1024 \* 1024) {

    error.value = 'Файл слишком большой (макс. 5MB)';

    return;

  }

  selectedFile.value = file;

  uploadProgress.value = 0;

  error.value = null; // Сбрасываем предыдущие ошибки

}

// Удаление текущего фото

async function removePhoto() {

  if (!currentPhoto.value) return;

  try {

    isUploadingPhoto.value = true;

    await api.patch(`/product/${props.productId}`, { remove\_photo: **true** });

    currentPhoto.value = null;

    form.value.photo = null;

  } catch (e) {

    error.value = 'Ошибка при удалении фото';

    console.error(e);

  } finally {

    isUploadingPhoto.value = false;

  }

}

// Загрузка нового фото

const uploadPhoto = async () => {

  if (!selectedFile.value) {

    error.value = 'Файл не выбран';

    return false;

  }

  // 1. Создаем FormData и добавляем файл

  const formData = new FormData();

  formData.append('photo', selectedFile.value, selectedFile.value.name); // Явно указываем имя файла

  // 2. Проверка содержимого FormData

  for (let [key, value] of formData.entries()) {

    console.log('FormData entry:', key, value);

  }

  try {

    // 3. Отправка PATCH-запроса

    const response = await axios({

      method: 'PATCH',

      url: `/product/${props.productId}/photo`,

      baseURL: 'http://secrets-of-the-east.ru/api',

      data: formData,

      headers: {

        'Content-Type': 'multipart/form-data',

        'Authorization': `Bearer ${localStorage.getItem('auth\_token')}`,

        'Accept': 'application/json'

      },

      onUploadProgress: progressEvent => {

        if (progressEvent.total) {

          uploadProgress.value = Math.round(

            (progressEvent.loaded \* 100) / progressEvent.total

          );

        }

      }

    });

    return response.data;

  } catch (e) {

    console.error('Full upload error:', {

      request: e.request,

      response: e.response?.data,

      config: e.config

    });

    error.value = e.response?.data?.message || 'Ошибка загрузки файла';

    return false;

  }

};

// Добавление новой пары цвет-размер

function addColorSize() {

  form.value.product\_color\_sizes.push({

    id: **null**,

    color\_id: **null**,

    size\_id: **null**,

    quantity: 0,

    isNewColor: **false**,

    new\_color\_name: '',

    new\_color\_hex: '#000000',

    isNewSize: **false**,

    new\_size\_name: '',

  });

}

// Удаление пары цвет-размер

function removeColorSize(index) {

  form.value.product\_color\_sizes.splice(index, 1);

}

// Общее количество товара

const totalQuantity = computed(() => {

  return form.value.product\_color\_sizes.reduce(

    (sum, pcs) => sum + (pcs.quantity || 0),

    0

  );

});

// Отправка формы

async function submitForm() {

  loadingSubmit.value = true;

  error.value = null;

  try {

    // Сначала загружаем фото, если оно было выбрано

    if (selectedFile.value) {

      const photoUploaded = await uploadPhoto();

      if (!photoUploaded) return;

    }

    // Группируем цвета и размеры

    const colorsGrouped = [];

    const mapColors = new Map();

    for (const pcs of form.value.product\_color\_sizes) {

      const colorKey = pcs.isNewColor

        ? `new\_${pcs.new\_color\_name}\_${pcs.new\_color\_hex}`

        : pcs.color\_id;

      if (!mapColors.has(colorKey)) {

        const colorObj = {sizes: []};

        if (pcs.isNewColor) {

          colorObj.new\_color\_name = pcs.new\_color\_name;

          colorObj.new\_color\_hex = pcs.new\_color\_hex;

        } else {

          colorObj.color\_id = pcs.color\_id;

        }

        mapColors.set(colorKey, colorObj);

      }

      const sizeObj = {};

      if (pcs.isNewSize) {

        sizeObj.new\_size\_name = pcs.new\_size\_name;

      } else {

        sizeObj.size\_id = pcs.size\_id;

      }

      sizeObj.quantity = pcs.quantity;

      mapColors.get(colorKey).sizes.push(sizeObj);

    }

    // Формируем payload

    const payload = {

      name: form.value.name,

      description: form.value.description,

      price: form.value.price,

      sex: form.value.sex,

      category\_id: form.value.category\_id,

      country\_id: form.value.country\_id,

      colors: Array.from(mapColors.values()),

    };

    await api.patch(`/product/${props.productId}`, payload);

    emits('updated');

    close();

  } catch (e) {

    error.value = 'Ошибка при сохранении товара';

    console.error(e);

  } finally {

    loadingSubmit.value = false;

  }

}

// Закрытие модального окна

function close() {

  emits('close');

}

// Загрузка данных при монтировании

onMounted(() => {

  loadReferenceData();

});

// Отслеживание изменения ID товара

watch(() => props.productId, (id) => {

  if (id) fetchProduct(id);

}, {immediate: **true**});

</script>

# ГЛАВА 6. ТЕСТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Название теста** | **Шаги** | **Ожидаемый результат** | **Актуальный результат** |
| **1** | Регистрация нового пользователя | Открыть раздел профиля, выбрать вкладку регистрации, заполнить форму, нажать кнопку регистрации | Пользователь регистрируется и автоматически авторизуется | Регистрация прошла успешно, пользователь вошёл в систему |
| **2** | Вход пользователя | Перейти в раздел профиля, ввести логин и пароль, нажать кнопку входа | Пользователь авторизуется и видит персональный интерфейс | Авторизация успешна, данные пользователя отображаются |
| **3** | |  | | --- | |  |  |  | | --- | | Добавление товара в корзину | | |  | | --- | |  |  |  | | --- | | Перейти на страницу товара, выбрать цвет и размер, нажать кнопку добавления в корзину | | Товар с выбранными параметрами добавляется в корзину | Товар добавлен в корзину и отображается в списке |
| **4** | |  | | --- | |  |  |  | | --- | | Оформление заказа | | Перейти в корзину, выбрать пункт выдачи, нажать кнопку подтверждения заказа | Создаётся заказ, отображается в заказах | Заказ оформлен, отображается в заказах пользователя |
| **5** | |  | | --- | |  |  |  | | --- | | Просмотр отзывов | | |  | | --- | |  |  |  | | --- | | Открыть страницу любого товара | | Отображается список опубликованных отзывов | Отзывы корректно отображаются |
| **6** | |  | | --- | |  |  |  | | --- | | Добавление отзыва пользователем с заказом | | |  | | --- | |  |  |  | | --- | | Войти в аккаунт, оформить заказ на товар, после доставки перейти на страницу товара и оставить отзыв | | Отзыв успешно добавляется и отображается в общем списке | Отзыв отображается на странице товара |
| **7** | |  | | --- | |  |  |  | | --- | | Попытка оставить отзыв без покупки | | Войти в аккаунт, перейти на страницу товара, не имея заказа на него, попытаться оставить отзыв | Система запрещает отправку отзыва | Появляется сообщение, что оставить отзыв можно только после покупки |
| **8** | Поиск товаров | |  | | --- | |  |  |  | | --- | | Ввести ключевое слово в строку поиска в шапке сайта | | Отображаются товары, соответствующие запросу | Поиск работает корректно, товары загружаются |
| **9** | |  | | --- | |  |  |  | | --- | | Применение фильтрации | | Открыть каталог, выбрать параметры фильтрации (цвет, цена, категория и т.д.) | Отображаются товары согласно выбранным параметрам | Товары фильтруются корректно |
| **10** | |  | | --- | |  |  |  | | --- | | Очистка фильтров | | |  | | --- | |  |  |  | | --- | | Очистить выбранные фильтры | | Отображается полный список товаров | Каталог возвращается в исходное состояние |
| **11** | Создание товара менеджером | Войти как менеджер, открыть раздел инструментов, нажать «Создать товар», заполнить форму | Товар создаётся и отображается в общем каталоге | Новый товар успешно создан и виден в списке |
| **12** | |  | | --- | |  |  |  | | --- | | Редактирование товара менеджером | | Войти как менеджер, открыть раздел товаров, нажать «Редактировать», изменить данные и сохранить | Изменённый товар отображается с новыми параметрами | Изменения сохранены, товар обновлён |
| **13** | Удаление отзыва менеджером | Войти как менеджер, открыть раздел отзывов, нажать «Удалить» рядом с нужным отзывом | Отзыв удаляется и больше не отображается | Отзыв удалён, интерфейс обновился |
| **14** | |  | | --- | |  |  |  | | --- | | Просмотр всех заказов менеджером | | Войти как менеджер, открыть раздел заказов | Отображаются все заказы пользователей | Все заказы отображаются, доступны для просмотра |
| **15** | Изменение статуса заказа менеджером | Открыть раздел заказов, выбрать нужный заказ, изменить его статус | Новый статус сохраняется и отображается | Статус успешно изменён |
| **16** | |  | | --- | |  |  |  | | --- | | Просмотр всех пользователей администратором | | Войти как администратор, открыть раздел пользователей | Отображается список всех зарегистрированных пользователей | Пользователи отображаются с возможностью редактирования |
| **17** | Управление пользователями | Перейти в раздел пользователей, выбрать нужного, нажать «Редактировать» или «Удалить» | Пользователь редактируется или удаляется из системы | Действия выполняются успешно, интерфейс обновляется |
| **18** | Создание категории администратором | Открыть раздел категорий, нажать «Создать», ввести название и сохранить | Новая категория появляется в списке фильтров каталога | Категория успешно создана и отображается |
| **19** | |  | | --- | |  |  |  | | --- | | Управление пунктами выдачи | | Перейти в раздел пунктов выдачи, создать или отредактировать пункт, сохранить изменения | Новый или обновлённый пункт отображается при оформлении заказа | Все изменения сохранены и отображаются |
| **20** | |  | | --- | |  |  |  | | --- | | Удаление менеджера | | |  | | --- | |  |  |  | | --- | | Перейти в раздел менеджеров, выбрать и удалить нужного | | Менеджер удаляется и теряет доступ к панели инструментов | Менеджер удалён, авторизация более невозможна |
| **21** | Просмотр профиля | Нажать на кнопку «Профиль» | Отображается информация о пользователе и список его заказов | Данные профиля корректно загружаются |
| **22** | Редактирование профиля | В разделе профиля изменить имя, email или телефон и сохранить | Данные обновляются и отображаются в новой версии | |  | | --- | | Данные успешно обновлены |  |  | | --- | |  | |
| **23** | |  | | --- | |  |  |  | | --- | | Выход из системы | | Нажать кнопку «Выйти» в профиле | Пользователь выходит из аккаунта, интерфейс возвращается в режим неавторизованного | Выход произведён, авторизация сброшена |

# ГЛАВА 7. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данный раздел содержит пошаговое руководство по работе с функционалом сайта «Тайны Востока». Здесь описаны основные действия, которые может выполнить пользователь: от регистрации и выбора товара до оформления заказа и взаимодействия с личным профилем. Руководство будет полезно как новым, так и постоянным пользователям сайта.

1. Вход и регистрация

1.1 Вход на сайт и регистрация

1. Для работы с корзиной и оформлением заказа необходимо быть авторизованным пользователем.
2. Авторизация и регистрация выполняются через кнопку «Профиль» в верхней части сайта.
3. Если у вас ещё нет аккаунта — зарегистрируйтесь, заполнив форму. После регистрации вы автоматически войдёте в систему.
4. Без авторизации добавление товара в корзину и оформление заказа невозможно.

2. Каталог товаров и фильтрация

2.1 Фильтры и товары

1. Страница каталога разделена на две части:

* Слева — блок фильтрации (категории, цвет, цена и др.).
* Справа — список товаров, соответствующих фильтрам.

1. Выберите нужные параметры фильтра для удобного поиска товаров.
2. Выберите интересующий товар для просмотра подробной информации.

2.2 Выбор товара

1. На странице товара сначала выберите цвет.
2. Затем выберите размер.
3. После выбора всех параметров станет доступна кнопка «Добавить в корзину».

3. Корзина и оформление заказа

3.1 Корзина

1. Корзина отображает товары с выбранными цветом и размером.
2. Если в корзине есть товары, это значит, что вы уже авторизованы.
3. В корзине можно изменить количество товаров или удалить их.
4. Итоговая сумма заказа отображается справа.
5. Для перехода к оформлению заказа нажмите кнопку «Оформить заказ».

3.2 Оформление заказа

1. При оформлении заказа выберите пункт выдачи — магазин из доступного списка.
2. Ввод адреса доставки не требуется, так как доставка происходит через выбранный магазин.
3. Проверьте список товаров и итоговую сумму.
4. Подтвердите заказ кнопкой «Подтвердить».

4. Профиль пользователя

4.1 Управление профилем

1. Перейдите в раздел «Профиль» через кнопку в верхнем меню.
2. Здесь вы можете просматривать и редактировать личные данные, такие как имя, email и телефон.
3. В разделе профиля также доступны ваши текущие и прошлые заказы.

5. Отзывы

5.1 Просмотр и добавление отзывов

1. Просматривать отзывы можно без авторизации.
2. Оставлять отзывы могут только авторизованные пользователи.
3. Чтобы оставить отзыв, перейдите в профиль, авторизуйтесь и на странице товара заполните форму отзыва.

Руководство пользователя сайта «Тайны Востока» призвано помочь вам быстро освоиться в интерфейсе и эффективно использовать все доступные функции.  
Внимательно следуя указанным шагам, вы сможете без труда находить и заказывать понравившиеся товары, управлять личным профилем, а также делиться впечатлениями в отзывах.

# ГЛАВА 8. РУКОВОДСТВО МЕНЕДЖЕРА

Данное руководство предназначено для менеджеров, работающих с административной частью сайта «Тайны Востока». Оно описывает основные действия, доступные менеджеру: вход в систему, управление товарами, редактирование информации, добавление цветов и размеров.

Руководство поможет быстро освоить интерфейс панели управления и эффективно выполнять свои обязанности.

1. Вход в систему

* Менеджер входит на сайт с учётными данными, выданными администратором.
* Самостоятельная регистрация отсутствует.

1. Доступ к панели инструментов

* После входа в систему в шапке сайта появляется кнопка «Инструменты».
* Нажмите на кнопку «Инструменты», чтобы открыть панель управления.

1. Создание товара с выбором цвета и размера

* В панели инструментов нажмите кнопку «Создать товар».
* Откроется форма создания товара, где необходимо заполнить основные параметры товара.
* В процессе создания товара у вас будет возможность:

1. выбрать существующий цвет из списка;
2. или создать новый цвет прямо в этой же форме, если подходящего нет;
3. аналогично — выбрать существующий размер или добавить новый размер в процессе создания.

* Это позволяет менеджеру быстро добавлять новые варианты цветов и размеров без выхода из формы создания товара.
* После заполнения всех данных и выбора (или создания) цвета и размера подтвердите создание товара.

1. Редактирование товаров

* В списке товаров нажмите кнопку «Редактировать» напротив нужного товара.
* В открывшейся форме редактирования можно изменить данные товара, включая цвета и размеры.
* Если необходимо, также можно создать новые цвета и размеры прямо из формы редактирования товара.
* Сохраните внесённые изменения.

1. Управление заказами и отзывами

* В панели инструментов доступен список всех заказов.
* Менеджер может просматривать каждый заказ и отслеживать его статус.
* Для управления нажмите на заказ — появится возможность изменить его статус (например, подтвердить или отменить).
* В товарах можно просматривать все отзывы пользователей.
* При необходимости любой отзыв можно удалить, например, если он нарушает правила площадки.

Менеджер играет важную роль в наполнении и актуализации контента на сайте «Тайны Востока».

Интерфейс панели управления интуитивно понятен, а возможность добавлять цвета и размеры прямо при создании или редактировании товара делает процесс быстрым и удобным.

# ГЛАВА 9. РУКОВОДСТВО АДМИНИСТРАТОРА

Администратор обладает максимальными правами на сайте «Тайны Востока» и может выполнять все действия менеджера, а также управлять пользователями, категориями, пунктами выдачи и другими сущностям системы.

Настоящее руководство содержит полное описание административных функций и поможет эффективно использовать все доступные возможности системы управления сайтом.

1. Вход в систему

* Администратор входит в систему с учётными данными, выданными при назначении.
* Самостоятельная регистрация администраторов недоступна.
* После авторизации в шапке сайта появляется кнопка «Инструменты» — основной доступ к панели управления.

1. Управление товарами (функционал менеджера)

* Администратор может создавать, редактировать и удалять товары.
* При создании или редактировании товара можно:
* выбрать или создать цвет;
* выбрать или создать размер.
* Все действия производятся через панель «Инструменты» → «Создать товар».
* Также доступен поиск товаров, фильтрация и просмотр подробностей.

1. Работа с заказами и отзывами (функционал менеджера)

* Администратор может просматривать все заказы клиентов.
* Для каждого заказа можно изменить статус (например, подтвердить, отменить и т.д.).
* Администратор может удалять отзывы, нарушающие правила.

1. Управление пользователями

* В панели «Инструменты» доступен раздел «Пользователи».
* Администратор может просматривать список всех пользователей, включая клиентов и менеджеров.
* Для каждого пользователя доступен подробный просмотр его профиля.
* Можно отредактировать данные пользователя или удалить его.
* Также доступна регистрация новых менеджеров с указанием всех необходимых данных.
* Менеджеров можно удалять, если они более не работают с системой.

1. Управление категориями

* Администратор может создавать новые категории одежды.

1. Управление пунктами выдачи

* Администратор имеет полный контроль над пунктами выдачи:
  + может добавить новый пункт;
  + изменить данные существующего;
  + удалить ненужный пункт.
* Эти пункты отображаются для клиента при оформлении заказа.

Администратор — ключевая роль в структуре управления сайтом «Тайны Востока».

Он не только выполняет функции менеджера, но и отвечает за контроль пользователей, настройки каталога, логистические точки и общее администрирование платформы.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта была успешно реализована REST API-система интернет-магазина восточной одежды "Тайны Востока" на базе фреймворка Laravel, а также разработан соответствующий веб-интерфейс с использованием Vue.js. Все поставленные цели и задачи, сформулированные во введении, были достигнуты в полном объеме.

1. Основные выполненные задачи проекта включают:

* Разработку REST API на Laravel, обеспечивающего:
* Полноценное управление товарами с системой фильтрации
* Функционал корзины покупок
* Оформление и обработку заказов
* Безопасную систему аутентификации

1. Создание веб-интерфейса на Vue.js, который предоставляет:

* Удобный каталог товаров с поиском и сортировкой
* Адаптивный дизайн для различных устройств
* Личный кабинет пользователя

1. Реализацию системы безопасности и управления доступом:

* Разграничение прав для администраторов, менеджеров и пользователей
* Защиту от распространенных веб-угроз
* Безопасное хранение данных

1. Организацию системы доставки:

* Интеграцию пунктов выдачи заказов
* Управление логистическими процессами

1. Разработку системы отзывов:

* Механизм добавления и модерации оценок
* Защиту от недобросовестных отзывов

Дополнительно были реализованы:

* Интеграция с платежной системой Stripe
* Оптимизация производительности базы данных

Все компоненты системы прошли комплексное тестирование и демонстрируют стабильную работу. Разработанное решение полностью соответствует современным требованиям к электронной коммерции и готово к промышленной эксплуатации.

Перспективы дальнейшего развития проекта включают:

* Создание мобильного приложения
* Интеграцию с системами складского учета
* Развитие рекомендательного сервиса
* Подключение дополнительных платежных систем

Таким образом, дипломный проект успешно завершен, все поставленные задачи выполнены, что подтверждает достижение заявленных целей и готовность системы к практическому применению.

# СПИСОК ЛИТЕРАТУРЫ

***Используемые интернет-источники:***

1. Руководство по PHP [Электронный ресурс]. – Режим доступа: свободный.

<https://metanit.com/php/>

1. Руководство по Laravel [Электронный ресурс]. – Режим доступа: свободный.

<https://laravel.com/>

1. Руководство по MySQL [Электронный ресурс]. – Режим доступа: свободный.

<https://metanit.com/sql/mysql/>

1. Список кодов состояния HTTP [Электронный ресурс]. – Режим доступа: свободный.

<https://en.wikipedia.org/wiki/List_of_HTTP_status_codes>

1. Vue.js – Прогрессивный JavaScript-фреймворк | Vue.js [Электронный ресурс]. – Режим доступа: свободный.

<https://ru.vuejs.org/>

1. Vue Router | Официальный маршрутизатор для Vue.js [Электронный ресурс]. – Режим доступа: свободный.

<https://vue-router-ru.netlify.app/>

1. What is Vuex? | Vuex [Электронный ресурс]. – Режим доступа: свободный.

<https://vuex.vuejs.org/>

1. VueUse [Электронный ресурс]. – Режим доступа: свободный.

<https://vueuse.org/>