ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

**«ТОМСКИЙ ЭКОНОМИКО-ПРОМЫШЛЕННЫЙ КОЛЛЕДЖ»**

К ЗАЩИТЕ ДОПУЩЕНЫ

Директор ОГБПОУ «ТЭПК»

\_\_\_\_\_\_\_\_\_\_ / О.Н. Пояркова

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

**ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНТЕРНЕТ-МАГАЗИНА ПО ПРОДАЖАМ МОТОТЕХНИКИ**

КОМПЛЕКСНЫЙ ДИПЛОМНЫЙ ПРОЕКТ

*Специальность 09.02.07 Информационные системы и программирование*

Выполнили:

Студенты IV курса группы 1912с

\_\_\_\_\_\_\_\_\_\_\_\_ / Ридт Д.А.

\_\_\_\_\_\_\_\_\_\_\_\_ / Бондаренко К.В.

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

Проверил:

Руководитель КДП

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

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

Томск 2025

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc201201424)

[**ТЕХНОЛОГИИ И ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ** 5](#_Toc201201425)

[**Используемые языки программирования и фреймворки** 5](#_Toc201201426)

[**Используемое программное обеспечение** 5](#_Toc201201427)

[**ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 8](#_Toc201201428)

[**Цели и задачи сайта** 8](#_Toc201201429)

[**ГЛАВА 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА** 10](#_Toc201201430)

[**Header** 10](#_Toc201201431)

[**Footer** 11](#_Toc201201432)

[**Главная страница** 12](#_Toc201201433)

[**Страница каталога** 17](#_Toc201201434)

[**Страница корзина** 19](#_Toc201201435)

[**Страница профиля** 21](#_Toc201201436)

[**Страница инфо** 22](#_Toc201201437)

[**Меню Вход/Регистрация** 23](#_Toc201201438)

[**Использование CSS для адаптации** 25](#_Toc201201439)

[**Использование @media для адаптации** 25](#_Toc201201440)

[**Адаптация под планшет** 25](#_Toc201201441)

[**ГЛАВА 2. РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА** 28](#_Toc201201442)

[**Разработка главной страницы сайта** 28](#_Toc201201443)

[**MainTop** 30](#_Toc201201444)

[**Footer** 33](#_Toc201201445)

[**Slider** 34](#_Toc201201446)

[**BlockMotorcycle** 35](#_Toc201201447)

[**Наши эндуро** 38](#_Toc201201448)

[**Страница Profile** 39](#_Toc201201449)

[**SliderMotorcycle** 41](#_Toc201201450)

[**ГЛАВА 3. ИНТЕГРАЦИЯ С БЭКЕНДОМ** 43](#_Toc201201451)

[**Основы взаимодействия с API** 43](#_Toc201201452)

[**ГЛАВА 4. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БД** 46](#_Toc201201453)

[**Концептуальная модель данных** 46](#_Toc201201454)

[**Нормализация данных** 48](#_Toc201201455)

[**Логическая модель данных** 49](#_Toc201201456)

[**Физическая модель данных** 49](#_Toc201201457)

[**ГЛАВА 5. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ API СЕРВИСА** 51](#_Toc201201458)

[**Описание проекта и задач** 51](#_Toc201201459)

[**Специфические требования к API** 52](#_Toc201201460)

[**Функционал пользователя** 54](#_Toc201201461)

[**Функционал Администратора** 57](#_Toc201201462)

[**ГЛАВА 6. ПРОЕКТИРОВАНИЕ И РАЗАРБОТКА API СЕРВИСА** 64](#_Toc201201463)

[**Установка и настройка фреймворка Laravel** 64](#_Toc201201464)

[**Создание миграций** 65](#_Toc201201465)

[**Генерация классов моделей** 69](#_Toc201201466)

[**Генерация классов контроллера** 78](#_Toc201201467)

[**Панель администратора Filament** 88](#_Toc201201468)

[**Маршрутизация** 120](#_Toc201201469)

[**ГЛАВА 7. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ** 122](#_Toc201201470)

[**Тесты функционала администратора** 129](#_Toc201201471)

[**Тесты функционала пользователя** 130](#_Toc201201472)

[**ЗАКЛЮЧЕНИЕ** 141](#_Toc201201473)

[**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ** 142](#_Toc201201474)

# **ВВЕДЕНИЕ**

Разработка сайтов является не маловажной и актуально задачей в наше время. Сайты многофункциональны, могут быть предоставлены как визитная карточка, портфолио, магазин, библиотека, просмотр видео-ресурсов и тд.

Самые немаловажные достоинства разработка сайтов заключается в том, что все сайты будут располагаться на сервере, а это значит, что к сайту будут иметь доступ все, кому нужна или будет интересна определённая информация. А сам сайт сможет постоянно пополняться и приобретать новые возможности и функционал, что не позволит потерять свою актуальность и востребованность.

А самое главное в актуальности создания сайта, состоит в том, что для быстрого и максимального распространения какой-либо информации до огромного количества людей, является способ именно создание собственного сайта. Веб-сайт позволяет представить любую информацию клиенту. С помощью него можно будет сообщать о новостях, об разных изменениях и также содержать отзывы разных клиентов.

***Описание предметной области***

Предметной областью является проект который представляет собой интернет-магазин по продаже мототехники. Основная цель проекта заключается в развитии сообщества мотолюбителей, увеличении продаж мототехники, а также предоставлении качественного сервиса для клиентов.

Цель дипломного проекта - проектирование и разработка интернет-магазина по продаже мототехники с использованием фреймворка Laravel и Vue3. Для достижения поставленной цели необходимо решить следующие задачи:

***Цель дипломного проекта*** - проектирование и разработка интернет-магазина по продажам мототехники используя фреймворк Laravel и Vue3.

Для достижения поставленной цели необходимо было решить следующие задачи:

1. Произвести анализ предметной области.
2. Написать техническое задание.
3. Создать адаптивный макет будущего сайта.
4. Спроектировать и разработать базу данных.
5. Спроектировать и разработать функционал информационного продукта.
6. Создать продающий продукт.
7. Написать пользовательский интерфейс
8. Разработать функционал сайта на стороне клиента.

# **ТЕХНОЛОГИИ И ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ**

## **Используемые языки программирования и фреймворки**

**PHP** (/pi:.eɪtʃ.pi:/ англ. PHP: Hypertext Preprocessor — «PHP: препроцессор гипертекста»; первоначально PHP/FI (Personal Home Page / Form Interpreter), а позже названный Personal Home Page Tools — «Инструменты для создания персональных веб-страниц») — C-подобный скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений. В настоящее время является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов.

**SQL** (МФА: [ˈɛsˈkjuˈɛl]; аббр. от англ. Structured Query Language — «язык структурированных запросов») — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

**Laravel** — бесплатный веб-фреймворк с открытым кодом, предназначенный для разработки с использованием архитектурной модели MVC (англ. Model View Controller — модель-представление-контроллер). Laravel выпущен под лицензией MIT.

**Vue 3 -** представляет современный прогрессивный фреймворк, написанный на языке TypeScript и предназначенный для создания веб-приложений на уровне клиента. Основная область применения данного фреймворка — это создание и организация пользовательского интерфейса.

Vue 3 является логическим продолжением предыдущей версии фреймвока - Vuejs 2, первый релиз которого вышел еще в феврале 2014 года. Его создателем является Эван Ю (Evan You), который до этого работал в Google над AngularJS. С тех пор фреймфорк динамично развивается. В сентябре 2020 года вышла текущая версия фреймворка - Vue 3, особенностью которой стало то, что эта версия полностью написана на языке TypeScript вместо JavaScript, который использовался ранее. Собственно, в данном руководстве рассматривается именно версия Vue 3.

## **Используемое программное обеспечение**

**Draw.io** — инструмент для создания диаграмм, блок-схем, интеллект-карт, бизнес-макетов, отношений сущностей, программных блоков и другого. Сервис распространяется на бесплатной основе с открытым исходным кодом. Draw.io обладает богатым набором функций для визуализации большинства задач пользователя.

При входе на сервис пользователь сразу попадает в рабочий интерфейс. У пользователя нет возможности для авторизации или регистрации, есть только опция выбора места для экспорта проекта. Процесс создания проекта выглядит следующим образом: пользователь перетаскивает из левой панели фигуры или элементы на рабочую поверхность, затем изменяет их — изменяет цвет, размер, шрифт текста, свойства фигуры (прозрачность, форма и т. д.). Draw.io позволяет отслеживать и восстанавливать изменения готовых проектов, импортировать и экспортировать в PDF, PNG, XML, VSDX, HTML, а также автоматически публиковать и делиться работами.

Инструмент работает с Google Диск, Google Workspace и Dropbox, глубоко интегрирован и удобен для работы с продуктами Confluence и Jira от Atlassian. Пользователи также могут работать с диаграммами в автономном режиме и сохранять их локально, используя настольное приложение для персональных компьютеров.

Инструмент позволяет создавать: графики, диаграммы, таблицы, презентации, блок-схемы, планы помещений, воронки продаж, ментальный карты, карты сайтов.

**Open Server Panel** — это портативная программная среда, созданная специально для веб-разработчиков с учётом их рекомендаций и пожеланий. Данный программный комплекс включает в себя тщательно подобранный набор серверного программного обеспечения, а также невероятно удобную и продуманную управляющую утилиту, которая обладает мощными возможностями по администрированию и настройке всех доступных компонентов.

OSPanel широко используется с целью разработки, отладки и тестирования веб-проектов, а также для предоставления веб-сервисов в локальных сетях.

**PhpStorm** — коммерческая кросс-платформенная интегрированная среда разработки для PHP. Разрабатывается компанией JetBrains на основе платформы IntelliJ IDEA.

Преимущества:

*Поддержка основных фреймворков.*

PhpStorm идеально подходит для работы с Symfony, Drupal, WordPress, Zend Framework, Laravel, Magento, Joomla!, CakePHP, Yii и другими фреймворками.

*Все, что нужно для PHP.*

PhpStorm глубоко анализирует структуру кода и действительно понимает ваш код, поддерживая все возможности языка PHP как в новых, так и в legacy-проектах. Редактор поддерживает автодополнение кода и рефакторинги, предотвращает ошибки на лету.

*Поддержка фронтенд-технологий.*

В PhpStorm вы можете работать с самыми современными технологиями: HTML 5, CSS, Sass, Less, Stylus, CoffeeScript, TypeScript, Emmet и JavaScript. При этом будут доступны рефакторинг, отладка и юнит-тестирование. Благодаря функции Live Edit все изменения можно тут же посмотреть в браузере.

*Встроенные инструменты для разработчиков.*

Однообразные задачи удобно выполнять прямо в PhpStorm. IDE интегрирована с системами контроля версий, поддерживает удаленное развертывание, базы данных и SQL, инструменты командной строки, Docker, Composer, REST-клиент и многие другие инструменты.

*PhpStorm = WebStorm + PHP + БД/SQL.*

PhpStorm включает в себя всю функциональность WebStorm, а также полноценную поддержку PHP, баз данных и SQL.

Анализируя плюсы и минусы каждого редактора кода, и имея опыт работы в каждом из них, для выполнения выпускной квалификационной работы мой выбор был сделан в сторону использования phpStorm, который идеально подходит для всех задач проекта.

**Postman** — это платформа API, позволяющая разработчикам проектировать, создавать, тестировать и повторять свои API. По состоянию на май 2024 года Postman сообщает о более чем 25 миллионах зарегистрированных пользователей и 75 000 открытых API, которые, по его словам, представляют собой крупнейший в мире публичный API-хаб. Штаб-квартира компании находится в Сан-Франциско и имеет офис в Бангалоре, где она была основана.

**Figma** — это веб-приложение и программное обеспечение для дизайнеров, которое позволяет создавать, редактировать и делиться макетами интерфейсов пользователя (UI) в режиме реального времени. Оно было разработано для упрощения процесса проектирования и прототипирования интерфейсов, а также для улучшения сотрудничества между дизайнерами и разработчиками.

**JetBrains WebStorm -** [интегрированная среда разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) на [JavaScript](https://ru.wikipedia.org/wiki/JavaScript), [CSS](https://ru.wikipedia.org/wiki/CSS) & [HTML](https://ru.wikipedia.org/wiki/HTML) от компании [JetBrains](https://ru.wikipedia.org/wiki/JetBrains), разработанная на основе платформы [IntelliJ IDEA](https://ru.wikipedia.org/wiki/IntelliJ_IDEA).

WebStorm обеспечивает [автодополнение](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%B4%D0%BE%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5), анализ кода на лету, [навигацию по коду](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B2%D0%B8%D0%B3%D0%B0%D1%86%D0%B8%D1%8F_%D0%BF%D0%BE_%D0%BA%D0%BE%D0%B4%D1%83), [рефакторинг](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3), [отладку](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D0%BA%D0%B0), и интеграцию с [системами управления версиями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8). Важным преимуществом [интегрированной среды разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) WebStorm является работа с проектами[[2]](https://ru.wikipedia.org/wiki/WebStorm#cite_note-_8899f4c9bd08a9d0-2) (в том числе, рефакторинг кода JavaScript, находящегося в разных файлах и папках проекта, а также вложенного в HTML). Поддерживается множественная вложенность (когда в документ на HTML вложен скрипт на Javascript, в который вложен другой код HTML, внутри которого вложен Javascript) — то есть в таких конструкциях поддерживается корректный рефакторинг.

# **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

## **Цели и задачи сайта**

1. Основная цель

Создать удобный, современный и высококонверсионный интернет-магазин мототехники, который обеспечит стабильный рост продаж за счет привлечения целевого трафика, повышения лояльности клиентов и эффективного продвижения новых моделей.

2. Ключевые задачи

2.1. Увеличение трафика

Для привлечения большего числа посетителей необходимо использовать комплексный маркетинговый подход, включающий таргетированную рекламу в социальных сетях (Instagram, Facebook, TikTok), поисковую рекламу (Google Ads), а также SEO-оптимизацию сайта под ключевые запросы, связанные с мототехникой. Дополнительно планируется запуск блога и YouTube-канала с полезным контентом: обзорами моделей, советами по выбору и обслуживанию мотоциклов. Целевой показатель – рост трафика на 30% в течение 3 месяцев.

2.2. Повышение конверсии

Сайт должен быть максимально удобным для покупателей, поэтому требуется провести аудит юзабилити и оптимизировать ключевые элементы: упростить процесс оформления заказа, добавить отзывы, фото- и видеоматериалы, а также внедрить чат-бота для оперативных консультаций. Это позволит увеличить конверсию в покупки на 20% к концу года.

2.3. Развитие лояльности клиентов

Для стимулирования повторных покупок будет разработана программа лояльности с бонусными баллами за каждую покупку, скидками за рекомендации и эксклюзивными предложениями для постоянных клиентов (ранний доступ к новинкам, специальные цены).

2.4. Продвижение новых моделей

Для успешного запуска новых товаров планируются онлайн-презентации с участием экспертов и блогеров, а также специальные условия для первых покупателей (скидки, бесплатная доставка, подарки). Информирование аудитории будет осуществляться через email-рассылки и push-уведомления. Цель – увеличение продаж новых моделей на 50% в первый месяц после старта акции.

3. Требования к сайту

3.1. Функциональность

Интерактивный и современный дизайн – удобная навигация, быстрый доступ к товарам, визуальная привлекательность.

Адаптивность – корректное отображение на всех устройствах (ПК, смартфоны, планшеты).

Обратная связь – онлайн-чат, форма заявки, возможность быстрой консультации.

3.2. Контент

Главная страница – рекламные баннеры, блоки с популярными товарами, удобный переход в категории.

Карточки товаров – фото, название, краткое описание, цена, кнопка покупки.

Дополнительные материалы – обзоры, видео, статьи с полезной информацией.

4. Анализ конкурентов

Основными конкурентами являются крупные маркетплейсы (Яндекс.Маркет, Motozilla), специализированные магазины (Мото24, МотоМир) и сообщества в соцсетях. Их слабые стороны – сложная навигация, отсутствие персонализации и ограниченные сервисы. Наш магазин сделает упор на:

Простое и быстрое оформление заказа.

Персональные рекомендации и бонусы.

Живое общение с клиентами (чат, консультации).

5. Особенности ниши

Растущий спрос на мототехнику среди профессионалов и любителей.

Онлайн-продажи как основной канал сбыта.

Важность сообщества – клиенты ценят экспертные мнения и обмен опытом.

# **ГЛАВА 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА**

Сам сайт состоит из 12 страниц, в них входит 2 модульных окна – регистрация и вход:

1. Header
2. Footer
3. Главная страница.
4. Страница каталога.
5. Страница товара – 6шт.
6. Корзина.
7. Профиль.
8. Страница информации.

## **Header**

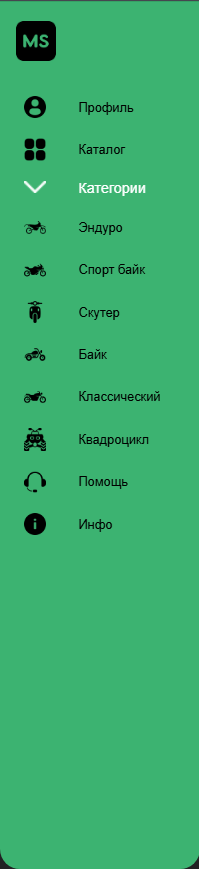


Рисунок 1 – Шапка сайта (Header)

Header сайта привязана к левой части сайта. Содержит Логотип, кнопки перехода в Профиль, Каталок и разные разделы каталога.

1. Раздел “Профиль” (при нажатии переход на страницу Профиля).
2. Раздел “Каталог” (при нажатии переход на страницу Каталога).
3. Выпадающий список “Категории”. При нажатии появляется список категорий товаров
4. Раздел “Помощь” (при нажатии можете написать администратору, если у вас возникли какие-то проблемы).
5. Раздел “Инфо” (при нажатии переход на страницу Инфо)
6. Логотип сайта, расположен вверху-слева.

* Логотип сайта – активная ссылка.
* При нажатии на логотип осуществляется переход на главную страницу сайта

Содержание выпадающего списка “Категории”

* Раздел “Эндуро”
* Раздел “Спорт байк”
* Раздел “Скутер”
* Раздел “Байк”
* Раздел “Классический”
* Раздел “Квадроцикл”

## **Footer**

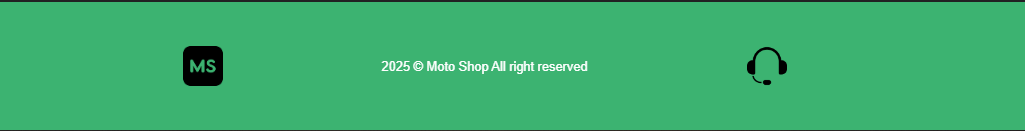


Рисунок 2 – Подвал сайта (Footer)

Footer содержит:

1. Логотип сайта, расположен слевва и центрирован по вертикали 
2. Политика конфиденциальности 
3. Иконка Помощь

## **Главная страница**

Макет страницы (см. рис 3):

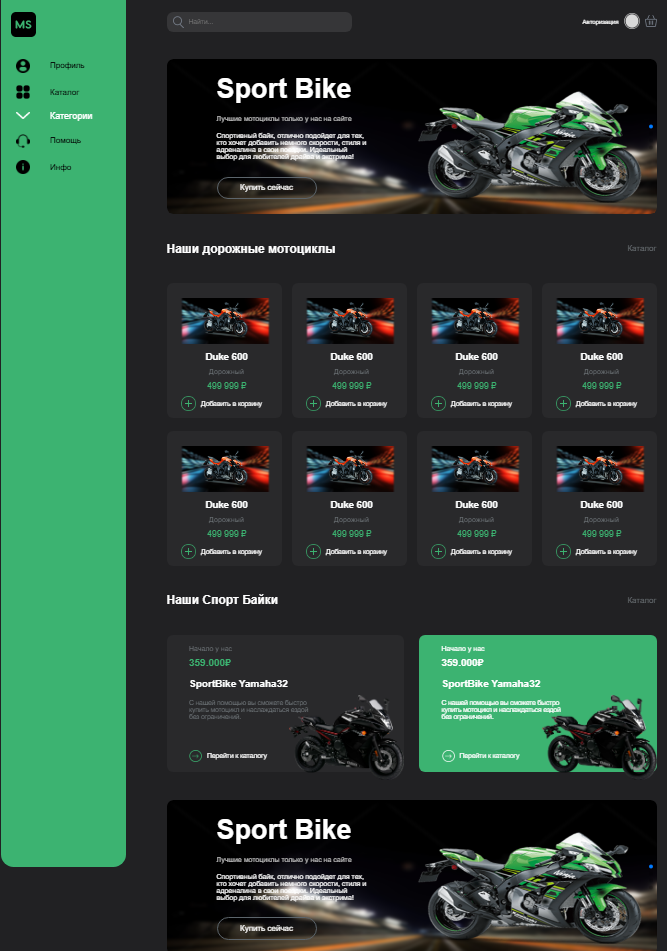


Рисунок 3- Макет «Главной страницы»

Блок «Рекламный баннер» (см. рис 4)



Рисунок 4 - Блок «Рекламный баннер»

Блок “Рекламный баннер” показывается актуальные акции

Блок “Рекламный баннер” содержит:

1. Картинку товара
2. Заголовок
3. Описание товара
4. Кнопку покупки

При наличии нескольких рекламных баннеров:

* Для переключения баннера с левой стороны находиться переключатель слайдеров

Блок “Наши дорожные мотоциклы ” (см. рис 5)



Рисунок 5 - Блок «Наше дорожные мотоциклы»

Блок “Наши дорожные мотоциклы” содержит карточку товаров

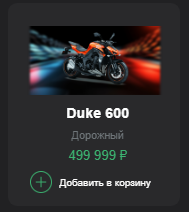


Рисунок 6 – Карточка товара

Карточка товара (см. рис 6) содержит:

1. Картинку
2. Название товара
3. Название класса мотоцикла
4. Стоимость
5. Кнопка добавить товар в корзину

При нажатии на карточку товара происходит переход на страницу описания товара

Страница товара (см. рис 7)

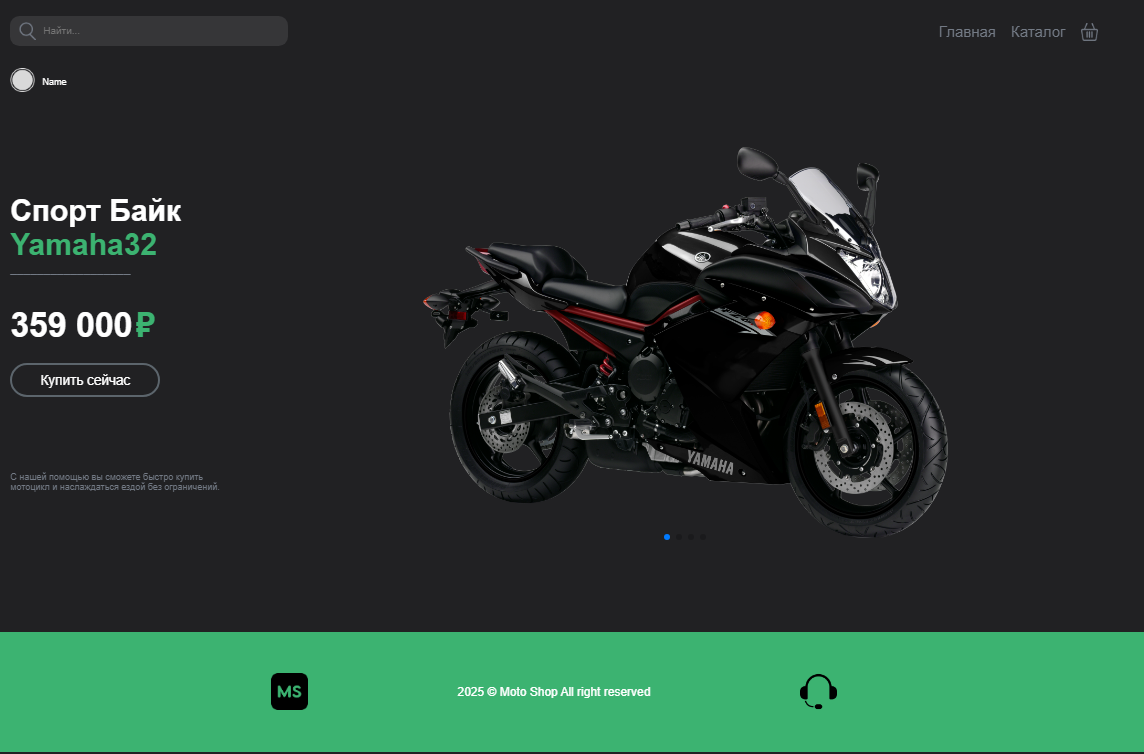


Рисунок 7 – Страница товара

Страница товара содержит:

* Название товара
* Стоимость товара
* Категория товара
* Кнопка покупки
* Слайдер с изображением товара
* Описание товара

Блок “Наши Спорт байки” (см. рис 8)

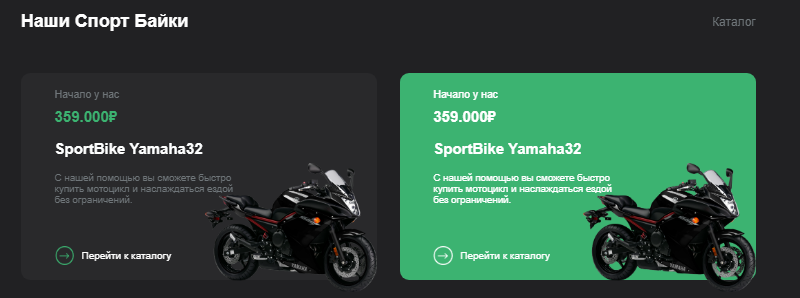


Рисунок 8 – Блок «Наши Спорт байки»

Блок “ Наши Спорт байки ” содержут карточку байков

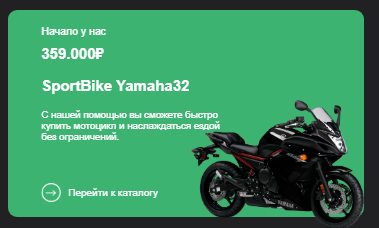


Рисунок 9 - Карточка байка

Карточка байка (см. рис 9) содержит:

1. Картинку товара
2. Стоимость товара
3. Название товара
4. Описание товара
5. Кнопку (при нажатии переход на страницу каталога)

Блок “Наши эндуро” (см. рис 10)



Рисунок 10 – Блок «Наши эндуро»

Блок “Наши эндуро” содержит карточку мотоциклов

1. Название мотоцикла
2. Стоимость
3. Описание
4. Кнопку добавления мотоцикла в корзину

## **Страница каталога**

Макет страницы (см. рис 12).



Рисунок 12 - Макет «Страницы каталога»

Навигационная панель и панель фильтрации товара (см. рис 13).

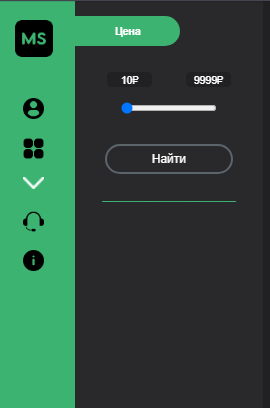


Рисунок 13 –Навигационная панель

Панель фильтрации товара содержит:

1. Выбор ценового диапазона (при помощи ползунка) 
2. Кнопка “Найти”

## **Страница корзина**

Макет страницы (см. рис 14).

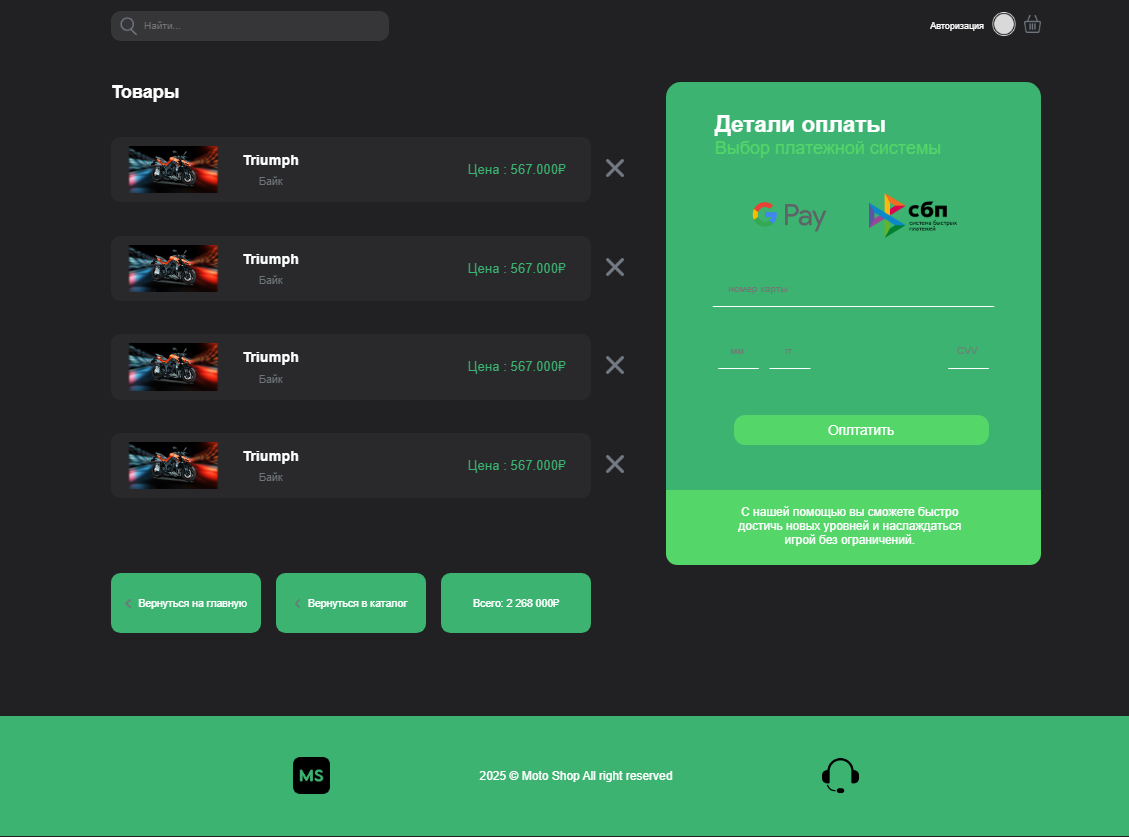


Рисунок 14 – Страница корзины

Здесь отображается товар который вы добавили в корзину.

Карточка товара (см. рис 15) содержит:

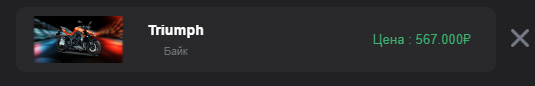


Рисунок 15 –Карточка товара

1. Картинку
2. Названия товара
3. Стоимость товара
4. Иконка удаления товара из корзины 

Блок “Детали оплаты” (см. рис 16) содержит:

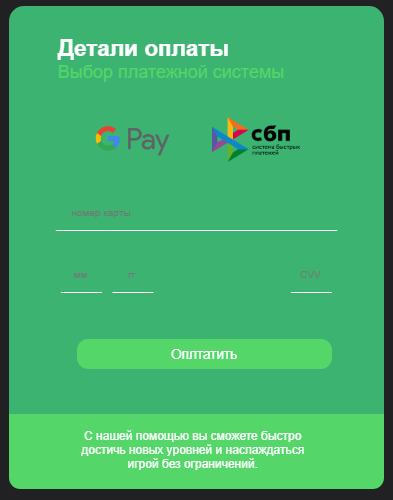


Рисунок 16 – Блок Оплаты

1. Заглавный текст
2. Картинки (выбора платежный системы)
3. Блок “Заполнения данных карты”
4. Кнопка “Оплатить”
5. Описание (Рекомендации)

Блок “Заполнения данных карты” (см. рис 17)



Рисунок 17 – Блок «Заполнения данных карты»

Блок заполнения данных карты содержит:

1. Поле “Номер карты”
2. Поле “дд”
3. Поле “гг”
4. Поле “cvv”

## **Страница профиля**

Макет страницы (см. рис 18).



Рисунок 18 - Макет «страница профиля»

Блок “Профиля” (см. рис 19)

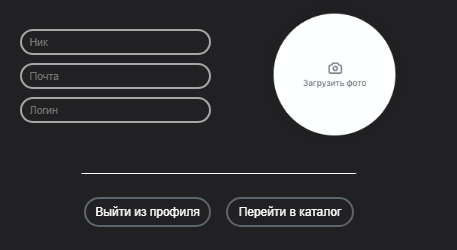


Рисунок 19 - Макет Блок «Профиля»

Блок “Профиля” содержит:

1. Поля для ввода данных профиля
2. Аватар профиля
3. Навигационные кнопки

## **Страница инфо**

Макет страницы (см. рис 20).

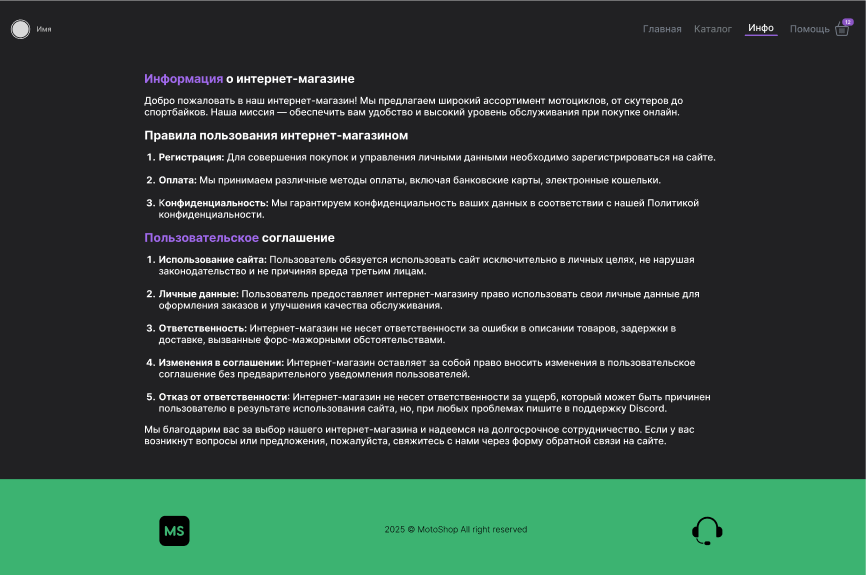


Рисунок 20 - Макет Блок «Инфо»

## **Меню Вход/Регистрация**

Макет меню “Регистрации” (см. рис 21)

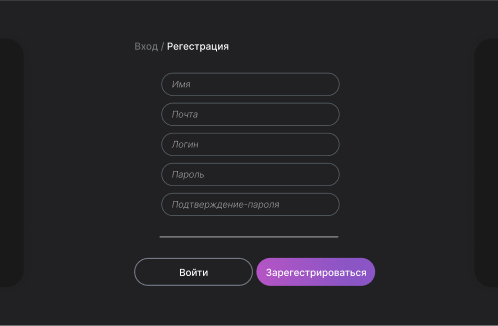


Рисунок 21 - Макет Блок «Регистрации»

Меню “Регистрации” содержит:

1. Поле “Ник”
2. Поле “Почта”
3. Поле “Логин”
4. Поле “Пароль”
5. Кнопки регистрации

Макет меню “Вход” (см. рис 22)



Рисунок 22 - Макет Блок «Вход»

Меню “Вход” содержит:

1. Поле “Логин”
2. Поле “Пароль”
3. Кнопка входа

АДАПТАЦИЯ

## **Использование CSS для адаптации**

CSS (Cascading Style Sheets) является ключевым инструментом в разработке адаптивных веб-сайтов. Он позволяет управлять стилем и макетом страницы, что особенно важно при адаптации к различным размерам экранов и ориентациям устройств.

## **Использование @media для адаптации**

@media — это директива, которая позволяет задавать разные стили для разных параметров экрана — ширины, высоты, ориентации и даже типа устройства.

## **Адаптация под планшет**

Главное меню (см. рис 23-24)

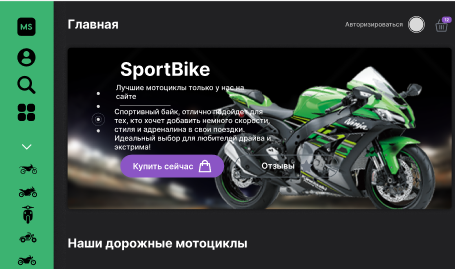


Рисунок 23 – Макет главной страницы

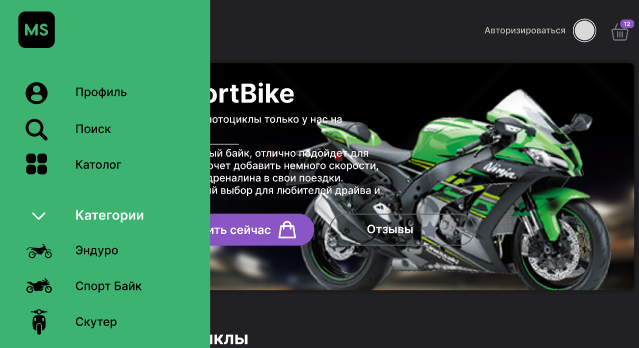


Рисунок 24 – Макет главной страницы

Страница Каталога (см. рис 25)

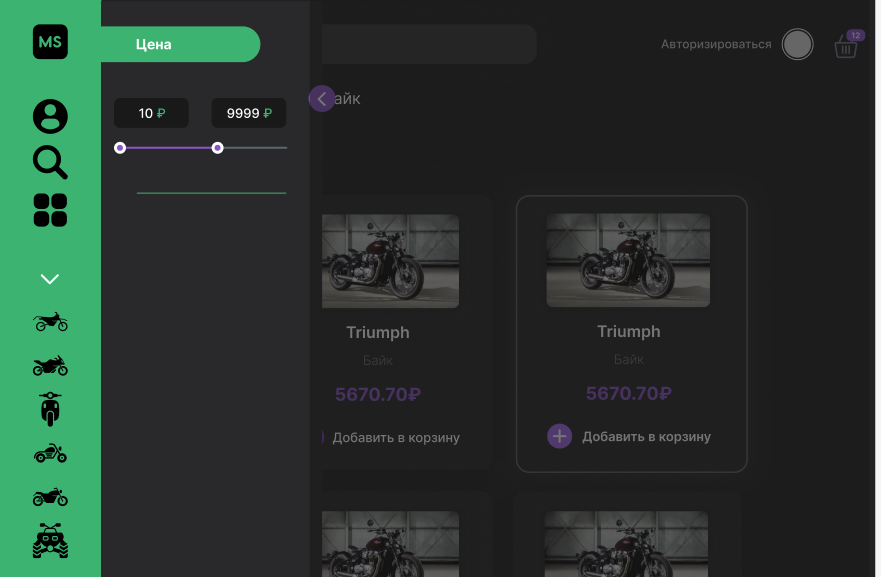


Рисунок 25 – Макет страницы каталога

Страница Корзина (см. рис 26)

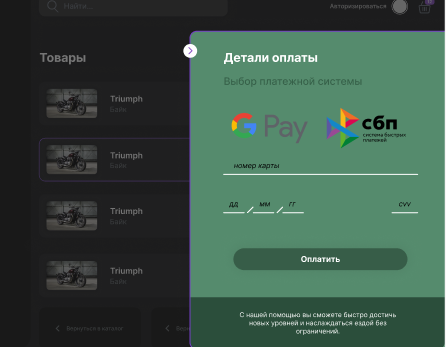


Рисунок 26 – Макет страницы корзины

**ГЛАВА 2. РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА**

## **Разработка главной страницы сайта**

Разработка главной страницы сайта является ключевым этапом в создании удобного и функционального интерфейса. Мы рассмотрим структуру и реализацию главной страницы, используя современные подходы в разработке пользовательского интерфейса.

1. **“Использование компонентов”:** В <script setup> импортируются различные компоненты, которые затем используются в шаблоне <template>. Это стандартный подход в Vue.js для создания сложных интерфейсов, где каждый компонент отвечает за определенную функциональность или часть пользовательского интерфейса.

2. **Шаблон <template>:**

- <div class="DashBoard">: Этот блок использует компонент Header, который, вероятно, отвечает за отображение основного панели управления или аналогичной функциональности.

- <div class="home">: В этом блоке размещены различные компоненты, такие как “MainTop”, “Slider”, “ BlockMotorcycle” и другие. Этот блок, вероятно, представляет собой главную часть страницы, где пользователь взаимодействует с контентом.

3. **Стили CSS:**

- ”home”: Стили для этого класса задают отступы, выравнивание и промежутки между элементами внутри блока. Это помогает организовать и настроить внешний вид компонентов внутри блока.

- @media (max-width: 880px): Медиа-запрос, который изменяет отступ слева для блока “.home” при ширине экрана до 880px. Это важно для адаптации интерфейса под различные устройства и размеры экрана.

- @media (max-width: 380px): Медиа-запрос, который изменяет отступ слева для блока “.home” при ширине экрана до 380px. Это важно для адаптации интерфейса под различные устройства и размеры экрана.

Листинг 1 – код главной страницы

<script setup>  
import MainTop from '@/components/MainTop.vue'  
import Slider from '@/components/Slider.vue'  
import SliderTwo from '@/components/SliderTwo.vue'  
import DashBoard from '@/components/DashBoard.vue'  
import EnduroCard from '@/components/EnduroCard.vue'  
import MotorcycleCard from '@/components/MotorcycleCard.vue'  
import BlockMotorcycle from '@/components/BlockMotorcycle.vue'  
</script>  
  
<template>  
 <div class="DashBoard">  
 <DashBoard/>  
 </div>  
 <div class="home">  
 <MainTop/>  
 <Slider/>  
 <BlockMotorcycle/>  
 <MotorcycleCard/>  
 <SliderTwo/>  
 <EnduroCard/>  
 </div>  
</template>  
  
<style scoped>  
.home {  
 padding-top: 25px;  
 display: flex;  
 flex-direction: column;  
 gap: 55px;  
 margin-left: 80px;  
}  
@media (max-width: 880px) {  
 .home{  
 margin-left: 20px;  
 }  
}  
@media (max-width: 380px) {  
 .home{  
 margin-left: 20px;  
 }  
}  
</style>

Эта структура и реализация позволяют создать главную страницу сайта, которая удобна для пользователей и легко масштабируется при необходимости.

**Header (Dashboard)**

1. **Использование `@vueuse/core` и реактивных переменных:**

- useWindowSize(): Этот хук из `@vueuse/core` используется для отслеживания ширины окна браузера.

- category и dash: Эти реактивные переменные управляют видимостью категорий и состоянием панели навигации.

2. **Функции управления состоянием:**

- showCategory(): Эта функция переключает состояние видимости категорий.

- showDash(): Эта функция переключает состояние видимости панели навигации.

3. **Шаблон <template>:**

- <div :class="{'back-color': dash}">: Этот div используется для отображения полупрозрачного фона, когда панель навигации раскрыта.

- <nav class="sidebar" :class="{'sidebar-width': dash }">: Этот <nav> элемент представляет сайдбар навигации, который изменяет ширину в зависимости от состояния dash.

- <transition name="fade">: Этот переход используется для плавного появления и исчезновения элементов при изменении их видимости.

Листинг 2 – код Header

<script setup>  
  
import { *useWindowSize* } from '@vueuse/core'  
import { *ref* } from 'vue'  
  
const { width } = *useWindowSize*()  
const category = *ref*(false)  
const dash = *ref*(false)  
  
function showCategory() {  
 category.value = !category.value  
}  
function showDash() {  
 dash.value = !dash.value  
}  
</script>

## **MainTop**

MainTop (см. рис 28)



Рисунок 28 – MainTop

1. **Импорты и использование библиотек:**

- ref` из Vue используется для создания реактивных переменных.

- useCookies из vue3-cookies для работы с куки.

- api из @/services/api/index.js для выполнения запросов к API.

- router для управления маршрутами.

2. **Реактивные переменные:**

- login и password для хранения введенных пользователем данных.

- modal для управления видимостью модального окна.

- isLocked для управления блокировкой скролла (хотя в коде это не используется).

- registration для переключения между режимами входа и регистрации.

- isAuth для проверки, авторизован ли пользователь.

3. **Функции управления состоянием:**

- showModal(): Функция, которая открывает или закрывает модальное окно и блокирует скролл. Если пользователь авторизован, перенаправляет на страницу профиля.

- switchReg(): Переключает состояние на регистрацию.

- switchLog(): Переключает состояние на вход.

- loginRequest(): Асинхронная функция для выполнения запроса на вход.

4. **Шаблон <template>:**

- <nav class="menu-top">: Верхняя навигационная панель с поиском и иконками.

- <Transition name="fade">: Используется для плавного появления и исчезновения модального окна.

- <div v-show="modal" class="modal">: Модальное окно, которое отображается при вызове функции `showModal()`.

5. **Стили CSS:**

-. fade-enter-active, fade-leave-active: Определяют анимацию перехода для элементов.

- .modal: Стили для модального окна, включая полупрозрачный фон.

- .button: Стили для кнопок входа и регистрации.

- Медиа-запросы для адаптации под разные размеры экрана.

Листинг 2 – основной код MainTop

<script>  
import { *ref* } from 'vue'  
*// import { useScrollLock } from '@vueuse/core';*import { api } from '@/services/api/index.js'  
import { *useCookies* } from 'vue3-cookies'  
import router from '@/router/index.js'  
  
export default {  
 setup() {  
 const { cookies } = *useCookies*()  
 const login = *ref*('') *// Реактивная переменная для хранения введенного логина* const password = *ref*('') *// Реактивная переменная для хранения введенного пароля* const modal = *ref*(false) *// Реактивная переменная для управления видимостью модального окна* const isLocked = *ref*(false) *// Реактивная переменная для управления блокировкой скролла* const registration = *ref*(false) *// Реактивная переменная для переключения между режимами входа и регистрации* const isAuth = *ref*(!!cookies.*isKey*('auth\_token')) *// Реактивная переменная для проверки авторизации пользователя  
// Функция для отображения или скрытия модального окна* const showModal = () => {  
 if (isAuth.value) {  
*// Если пользователь авторизован, перенаправляем на страницу профиля* return router.go('/profile')  
 }  
 modal.value = !modal.value *// Переключаем видимость модального окна* isLocked.value = !isLocked.value *// Переключаем состояние блокировки скролла  
  
// Обновляем состояние блокировки скролла  
// useScrollLock(document.body, isLocked.value);* }  
*// Функция для переключения в режим регистрации* const switchReg = () => {  
 registration.value = true  
 }  
  
*// Функция для переключения в режим входа* const switchLog = () => {  
 registration.value = false  
 }  
*// Асинхронная функция для выполнения запроса на вход* const loginRequest = async () => await api.auth.login({login: login.value, password: password.value})  
  
*// Возвращаем объект с переменными и функциями, которые будут доступны в шаблоне* return {  
 isAuth,  
 login,  
 password,  
 modal,  
 registration,  
 switchLog,  
 showModal,  
 switchReg,  
 loginRequest  
 }  
 }  
}  
</script>  
<template>  
 <nav class="menu-top">  
 <header>  
 <div class="search\_main">  
 <img src="../../image/Search.png" alt="#" />  
 <input class="text\_search" type="search" placeholder="Найти...">  
 </div>  
 <div class="menu-icons">  
 <span @click="showModal" class="text\_nickname">{{ isAuth ? 'Профиль' : 'Авторизация' }}</span>  
 <router-link to="profile">  
 <a><img src="../../image/Avatar.png" alt="profile" /></a>  
 </router-link>  
 <router-link to="Basket">  
 <img src="../../image/Cart.png" alt="/Basket" />  
 </router-link>  
 </div>  
 </header>  
 </nav>  
 <Transition name="fade">  
 <div v-show="modal" class="modal">  
 <div class="modal-content">  
 <div class="modal-main">  
 <span @click="showModal" class="cross-active">  
 <img src="../../image/Cross-active.png" alt="active" />  
 </span>  
 <div class="login-box">  
 <div class="text-log\_container">  
 <div class="text-log" :class="{'white' : !registration}" @click="switchLog">Вход</div>  
 <div class="white">/</div>  
 <div class="text-log" :class="{'white' : registration}" @click="switchReg">Регистрация</div>  
 </div>  
 <div class="text-write\_container" v-show="!registration">  
 <div class="text-write">  
 <input class="text\_search\_log" type="search" v-model="login" placeholder="Логин">  
 </div>  
 <div class="text-write\_2">  
 <input class="text\_search\_log" type="search" v-model="password" placeholder="Пароль">  
 </div>  
 </div>  
 <div class="text-write\_container" v-show="registration">  
 <div class="text-write">  
 <input class="text\_search\_log" type="search" placeholder="Ник">  
 </div>  
 <div class="text-write\_2">  
 <input class="text\_search\_log" type="search" placeholder="Почта">  
 </div>  
 <div class="text-write">  
 <input class="text\_search\_log" type="search" placeholder="Логин">  
 </div>  
 <div class="text-write\_2">  
 <input class="text\_search\_log" type="search" placeholder="Пароль"  
 required  
 v-model="password" />  
 </div>  
 </div>  
 <div class="button-log">  
 <button class="button" @click="loginRequest">Войти</button>  
 <button class="button">Регистрация</button>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </Transition>  
</template>

## **Footer**

В вашем Vue компоненте реализован нижний колонтитул (footer) с логотипом, текстом и иконкой Помощь. Вот несколько комментариев по вашему коду:

1. **Компонент нижнего колонтитула:**

- Вы используете <div> для создания структуры нижнего колонтитула, который включает в себя логотип, текст с правом авторских прав и иконку Помощь.

2. **Дизайн и стили:**

- Вы используете CSS в стиле scoped для стилизации компонента. Это хорошая практика, так как стили будут применяться только к этому компоненту и не повлияют на другие части вашего приложения.

- Нижний колонтитул имеет фиксированную высоту и цвет фона, который задается через CSS переменную `--Dash-Board`.

- Элементы внутри `.footer\_content` расположены горизонтально и равномерно распределены с помощью `justify-content: space-between`.

3. **Контент:**

- Текст "2025 © Moto Shop All right reserved " указывает на год, авторские права и название компании. Убедитесь, что эта информация актуальна и корректна.

Листинг 3 – основной код Footer

<script setup>  
</script>  
<template>  
<div class="footer">  
 <div class="footer-container container">  
 <div class="footer\_content">  
 <img src="../../image/montserratt.png" alt=""/>  
 <span>2025 © Moto Shop All right reserved</span>  
 <img src="../../image/supporticon.png" width="50" height="50"/>  
 </div>  
 </div>  
</div>  
</template>  
  
<style scoped>  
.footer{  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 height: 160px;  
 background: var(--Dash-Board);  
 margin-top: 110px;  
}  
.footer-container {  
 width: 100%;  
}  
.footer\_content{  
 margin-inline: auto;  
 width: 100%;  
 max-width: 755px;  
 display: flex;  
 justify-content: space-between;  
 align-items: center;  
}  
  
span{  
 color: white;  
}  
</style>

## **Slider**

В вашем Vue компоненте используется библиотека Swiper для создания слайдера. Вот несколько комментариев и уточнений по вашему коду:

1. **Использование Swiper:**

- Вы успешно импортировали компоненты Swiper и SwiperSlide, а также модуль Pagination. Это позволяет вам создавать адаптивные и функциональные слайдеры.

2. **Настройка слайдера:**

- Слайдер настроен с вертикальным направлением (`direction: 'vertical'`), что может быть полезно для специфических дизайнерских решений.

- Параметр `pagination.clickable: true` позволяет пользователям переключать слайды, нажимая на точки пагинации.

3. **Стили:**

- Стили для `.swiper` и `.swiper-slide` настроены для адаптации под разные экраны с помощью медиа-запроса. Это хорошая практика для создания адаптивного дизайна.

- Стили для `.button` включают hover-эффект, который изменяет цвет фона и границы, что улучшает взаимодействие с пользователем.

Листинг 4 – код Slider

<script>  
import { Swiper, SwiperSlide } from 'swiper/vue';  
  
import 'swiper/css';  
  
import 'swiper/css/pagination';  
  
import { Pagination } from 'swiper/modules';  
  
export default {  
 components: {  
 Swiper,  
 SwiperSlide,  
 },  
 setup() {  
 return {  
 modules: [Pagination],  
 };  
 },  
};  
  
</script>

## **BlockMotorcycle**

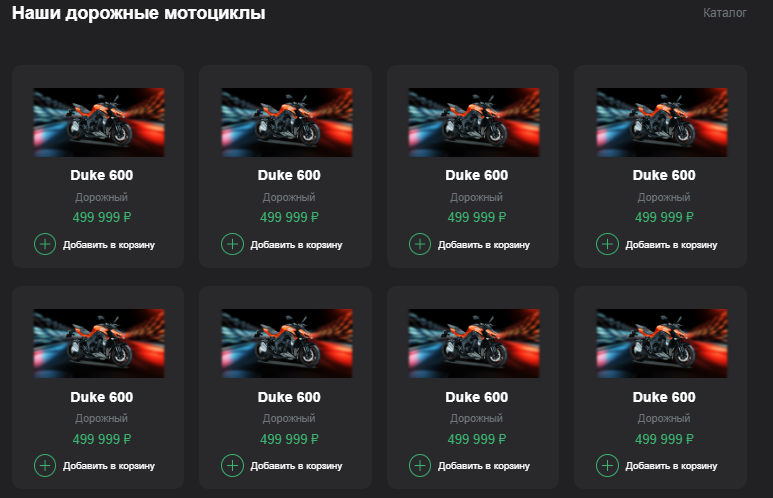


Рисунок 29 – Блок «BlockMotorcycle»

1. **Загрузка данных:**

- Функция `fetchProduct` использует `fetch` для запроса данных от API. URL для запроса берется из переменной окружения `VITE\_API\_URL`.

- Вы используете заголовок `"ngrok-skip-browser-warning"` для подавления предупреждения от ngrok, если вы используете его для тестирования.

- После получения данных, вы выбираете первые 8 элементов из ответа (`response.data.slice(0, 8)`), что может быть полезно для демонстрации или ограничения количества отображаемых элементов.

2. **Обработка ошибок:**

- В случае ошибки при загрузке данных, вы устанавливаете `data.value` равным объекту ошибки. Это может быть полезно для отладки, но обычно лучше обрабатывать ошибки более детально, например, отображать сообщение пользователю.

3. **Отображение данных:**

- Вы используете `v-for` для создания карточек товаров на основе данных из `data`. Каждая карточка содержит изображение, название, описание и цену товара.

- Каждая карточка также содержит кнопку "Добавить в корзину", которая вызывает функцию `addToCart` при нажатии. Эта функция в текущей реализации только логирует товар в консоль.

4. **Тестирование и отладка:**

- Убедитесь, что функция `addToCart` корректно обрабатывает добавление товаров в корзину. В текущей реализации она только логирует товар, но в реальном приложении вам нужно будет реализовать логику добавления товара в корзину.

Этот код является хорошей основой для создания компонента, который загружает и отображает данные из API, а также позволяет пользователям взаимодействовать с этими данными.

Листинг 5 – код BlockMotorcycle

<template>  
 <div class="overview">  
 <div class="nav-motorcycle">  
 <h2>Наши дорожные мотоциклы</h2>  
 </div>  
 <span class="cat">Каталог</span>  
 </div>  
 <div class="motorcycle-grid">  
 <div v-for="motorcycle in localMotorcycles" :key="motorcycle.id" class="motorcycle-card">  
 <img :src="motorcycle.image" alt="" class="motorcycle-image">  
 <h2 class="motorcycle-name">{{ motorcycle.title }}</h2>  
 <p class="motorcycle-description">{{ motorcycle.comment }}</p>  
 <p class="motorcycle-price">{{ motorcycle.price }}</p>  
 <div class="down-cart">  
 <img src="../../image/Add.png" alt="" />  
 <button class="add-to-cart" @click="addToCart(motorcycle)">  
 Добавить в корзину  
 </button>  
 </div>  
 </div>  
 </div>  
</template>

**MotorcycleSportCard**

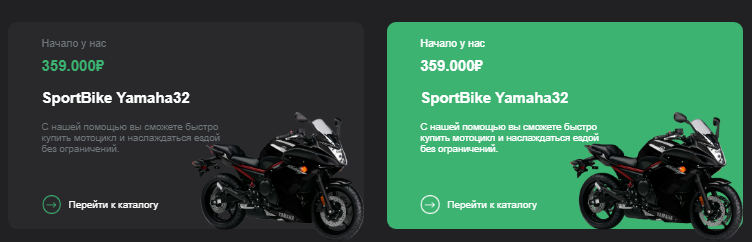
****

Рисунок 30 – Блок « MotorcycleSportCard»

1. **Пропсы:**

- Вы определили пропсы price, title, description и pink. Эти пропсы позволяют передавать данные о товаре и параметр, влияющий на стиль компонента.

2. **Динамические классы:**

- Вы используете синтаксис динамических классов для применения разных стилей в зависимости от значения пропса pink. Если pink истинно, элементы будут иметь класс white, что, вероятно, меняет цвет текста на белый.

3. **Изображения и стили:**

- Изображения в компоненте меняются в зависимости от значения pink. Если pink истинно, используется одно изображение, если ложно — другое. Это позволяет адаптировать внешний вид компонента под текущие настройки.

4. **Ссылка на маршрут:**

- Вы используете <router-link> для создания ссылки на другой маршрут (в данном случае "Catalog"). Это хорошая практика для создания кликабельных ссылок, которые переключают маршруты в Vue Router.

Листинг 6 – код MotorcycleSportCard

<script>  
export default {  
 props: {  
 price: *Number*,  
 title: *String*,  
 description: *String*,  
 pink: *Boolean*,  
 }  
}  
</script>  
  
<template>  
 <div class="motorcycle-card" :class="{pink}">  
 <img class="motorcycle-card-images" src="../../image/SportBikeWeapon.png" alt=""/>  
 <div class="motorcycle-card\_\_content">  
 <div class="Opis">  
 <span :class="{'white':pink}">Начало у нас</span>  
 <p class="motorcycle-price" :class="{'white':pink}">{{price}}₽</p>  
 <h2 class="motorcycle-name" :class="{'white':pink}">{{title}}</h2>  
 <p class="motorcycle-description" :class="{'white':pink}">{{description}}</p>  
 </div>  
 <div class="down-cart">  
 <img v-if="!pink" src="../../image/Arrow-krug.png" alt=""/>  
 <img v-else src="../../image/Button\_White.png" alt=""/>  
 <router-link to="Catalog">  
 <button class="add-to-cart">Перейти к каталогу</button>  
 </router-link>  
 </div>  
 </div>  
 </div>  
</template>

## **Наши эндуро**

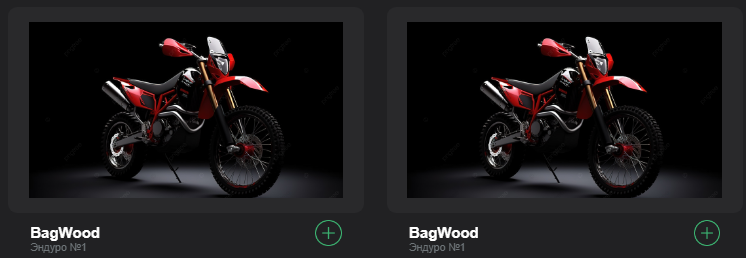


Рисунок 31 – Блок «OurCloth»

В вашем HTML и CSS коде представлен шаблон для веб-страницы, которая включает в себя навигационное меню, каталог одежды и стили для этих элементов. Вот некоторые комментарии и уточнения по коду:

1. **Структура:**

- <template> используется для определения шаблона, который может быть использован в JavaScript фреймворках, таких как Vue.js или Angular.

- Внутри <template> у вас есть разделы для навигации (nav-tish), общего представления (`overview`) и карточек товаров (`carts`).

2. **Стили:**

- Используется scoped для ограничения стилей только этим компонентом, что предотвращает их влияние на другие части страницы.

- Стили для кнопок (btn) включают hover эффект, который изменяет цвет фона и границы при наведении курсора.

- Карточки товаров (weapon-card и weapon-card2) имеют одинаковые стили, что указывает на возможную повторяемость кода. Можно объединить их в один класс, если они идентичны.

- Вы используете CSS переменные (var(--Button), var(--Card), var(--Title-h2)) для цветов, что упрощает управление стилями и поддержку кода.

Листинг 7 – код EnduroCard

<script setup>  
  
</script>  
  
<template>  
 <div class="overview">  
 <div class="nav-tish">  
 <h2>Наши эндуро</h2>  
 </div>  
 <span class="cat">Каталог</span>  
 </div>  
 <div class="carts">  
 <div class="carts\_\_content">  
 <div class="motorcycle-card">  
 <img src="../../image/Enduro.png" alt="" />  
 </div>  
 <div class="description">  
 <div class="text\_\_content">  
 <h2>BagWood</h2>  
 <p>Эндуро №1</p>  
 </div>  
 <div class="img\_\_content">  
 <img src="../../image/Add.png" alt="" />  
 </div>  
 </div>  
 </div>  
 <div class="carts\_\_content">  
 <div class="motorcycle-card2">  
 <img src="../../image/Enduro.png" alt="" />  
 </div>  
 <div class="description">  
 <div class="text\_\_content">  
 <h2>BagWood</h2>  
 <p>Эндуро №1</p>  
 </div>  
 <div class="img\_\_content">  
 <img src="../../image/Add.png" alt="" />  
 </div>  
 </div>  
 </div>  
 </div>  
</template>

## **Страница Profile**

1. **Подключение компонентов и загрузка данных:**

- Вы импортируете компоненты MainTop и DashBoard, которые, вероятно, используются для отображения верхней части страницы и панели управления соответственно.

- Вы используете ref` для создания реактивной переменной profile и загружаете данные профиля с помощью функции profileRequest из вашего API.

2. **Отображение данных профиля:**

- Вы используете директиву v-model для двустороннего связывания данных в полях ввода с данными профиля. Это позволяет обновлять данные профиля в реальном времени при изменении данных в полях.

3. **Кнопки действий:**

- Вы используете <router-link> для создания ссылки на маршрут "back\_catalog", что позволяет перейти в каталог с профиля пользователя.

7. **Функциональность:**

- Убедитесь, что функция выхода из профиля корректно реализована и обеспечивает безопасное разлогирование пользователя.

Листинг 8 – код страницы Profile

script setup>  
import MainTop from '@/components/MainTop.vue'  
import DashBoard from '@/components/DashBoard.vue'  
import { *ref* } from 'vue'  
import { api } from '@/services/api/index.js'  
  
const { profile: profileRequest } = api.auth  
let profile = *ref*({})  
profileRequest().then(data => profile.value = data)  
</script>  
  
<template>  
<DashBoard/>  
 <div class="home">  
<MainTop/>  
 <h2>Профиль</h2>  
 <div class="img\_ava\_container">  
 <div class="profile">  
 <div class="log-ava">  
 <div class="text-write\_container\_profile">  
 <div class="text-write">  
 <input class="text\_search\_log\_ava" type="search" v-model="profile.name" placeholder="Ник">  
 </div>  
 <div class="text-write\_2">  
 <input class="text\_search\_log\_ava" type="search" v-model="profile.email" placeholder="Почта">  
 </div>  
 <div class="text-write">  
 <input class="text\_search\_log\_ava" type="search" v-model="profile.login" placeholder="Логин">  
 </div>  
 </div>  
 <div class="avatar">  
 <img src="../../image/Ava.png" height="222" width="222" />  
 </div>  
 </div>  
 <div class="line-white">\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_</div>  
 <div class="button-log-profile">  
 <button class="button\_ava">Выйти из профиля</button>  
 <router-link to="back\_catalog">  
 <button class="button\_ava">Перейти в каталог</button>  
 </router-link>  
 </div>  
 </div>  
 <div class="img\_ava">  
 <img src="../../image/pngwing.com.png" alt="#" />  
 </div>  
 </div>  
 </div>  
</template>

## **SliderMotorcycle**

1. **HTML (Vue Template):**

- <template> содержит разметку для слайдера.

- <swiper> компонент используется для основного контейнера слайдера.

- :spaceBetween="30" устанавливает пространство между слайдами в 30 пикселей.

- :pagination="{ clickable: true }" добавляет кликабельную пагинацию, которую можно использовать для переключения слайдов.

- <swiper-slide> компоненты содержат изображения, которые будут отображаться в слайдере.

2. **JavaScript (Vue Script):**

- import { Swiper, SwiperSlide } from 'swiper/vue'; импортирует Vue компоненты Swiper и SwiperSlide.

- import swiper/css/pagination'; импортирует необходимые стили для пагинации.

- import { Pagination } from 'swiper/modules'; импортирует модуль пагинации, который необходим для работы пагинации в слайдере.

- В компоненте Vue components объявляет использование Swiper и SwiperSlide.

- В setup() функции возвращается массив `modules`, который включает в себя модуль пагинации. Это необходимо для того, чтобы Swiper знал, какие дополнительные функции включить.

Листинг 9 – код страницы SliderMotorcycle

<template>  
 <div class="swiper\_moto\_container">  
 <swiper  
 :spaceBetween="30"  
 :pagination="{  
 clickable: true,  
 }"  
 :modules="modules"  
 class="mySwiper">  
  
 <swiper-slide class="swiper\_moto"><img src="../../image/SportBikeWeapon.png" alt=""/></swiper-slide>  
 <swiper-slide class=" swiper\_moto "><img src="../../image/SportBikeWeapon.png" alt=""/></swiper-slide>  
 <swiper-slide class=" swiper\_moto "><img src="../../image/SportBikeWeapon.png" alt=""/></swiper-slide>  
 <swiper-slide class=" swiper\_moto "><img src="../../image/SportBikeWeapon.png" alt=""/></swiper-slide>  
 </swiper>  
 </div>  
</template>  
<script>  
*// Import Swiper Vue.js components*import { Swiper, SwiperSlide } from 'swiper/vue';  
  
*// Import Swiper styles*import 'swiper/css/pagination';  
  
  
*// import required modules*import { Pagination } from 'swiper/modules';  
  
export default {  
 components: {  
 Swiper,  
 SwiperSlide,  
 },  
 setup() {  
 return {  
 modules: [Pagination],  
 };  
 },  
};  
</script>

# **ГЛАВА 3. ИНТЕГРАЦИЯ С БЭКЕНДОМ**

В современном веб-разработчике взаимодействие с бэкендом через API является неотъемлемой частью создания функциональных и интерактивных приложений. API (Application Programming Interface) позволяет фронтенд-приложению обмениваться данными с сервером, выполнять различные операции и получать необходимую информацию. В этой главе мы рассмотрим, как организовано взаимодействие с бэкендом в контексте нашего проекта, используя библиотеку *axios* для создания HTTP-запросов и структурированные модули API для удобства и надежности.

## **Основы взаимодействия с API**

API предоставляет набор эндпоинтов, каждый из которых отвечает за определенные операции или данные. В нашем случае, API разделено на несколько частей, каждая из которых соответствует различным аспектам приложения: квесты, акции, услуги, праздники и т.д. Это разделение помогает упростить управление запросами и ответами, а также улучшить читаемость и поддержку кода.

Листинг 10 – код подключения API

VITE\_API\_URL=https://aabb-5-77-15-55.ngrok-free.app

Листинг 11 – код подключение карточек товара

<template>  
 <div class="overview">  
 <div class="icon\_menu">  
 <span class="cat">Главная</span>  
 <div class="white">/</div>  
 <span class="cat">Каталог</span>  
 <div class="white">/</div>  
 <span class="white">Мотоциклы</span>  
 </div>  
 <h2>Наши Байки</h2>  
 <div class="weapon-grid">  
 <div v-for="motorcycle in data" :key="motorcycle.id" class="weapon-card">  
 <img :src="motorcycle.image" alt="" class="weapon-image">  
 <h2 class="weapon-name">{{ motorcycle.title }}</h2>  
 <p class="weapon-description">{{ motorcycle.comment }}</p>  
 <p class="weapon-price">{{ motorcycle.price }}</p>  
 <div class="down-cart">  
 <img src="../../image/Add.png" alt="" />  
 <button class="add-to-cart" @click="addToCart(motorcycle)">  
 Добавить в корзину  
 </button>  
 </div>  
 </div>  
 </div>  
 </div>  
</template>  
  
<script setup>  
import { *ref* } from 'vue'  
  
const data = *ref*([]) *// Пустой массив (данные будут загружаться с API)*async function fetchMotorcycles() {  
 try {  
 const res = await *fetch*(import.meta.env.VITE\_API\_URL + "/api-motoshop.ru", {  
 method: "GET",  
 headers: {  
 "ngrok-skip-browser-warning": "69420",  
 }  
 })  
 const response = await res.json()  
 data.value = response.data.slice(0, 8) *// Ограничиваем 8 элементами* } catch (err) {  
 console.error("Ошибка загрузки данных:", err)  
 }  
}  
  
function addToCart(motorcycle) {  
 console.log("Добавлено в корзину:", motorcycle)  
}  
  
*// Загружаем данные при монтировании компонента*fetchMotorcycles()  
</script>  
  
<style scoped>  
h2 {  
 color: white;  
}  
  
.overview {}  
  
.cat {  
 color: var(--Title-h2);  
}  
  
.icon\_menu {  
 display: flex;  
 align-items: center;  
 gap: 15px;  
 padding-bottom: 35px;  
}  
  
.weapon-grid {  
 display: grid;  
 grid-template-columns: repeat(4, 225px);  
 gap: 25px;  
 padding-top: 40px;  
}  
  
.weapon-card {  
 height: 270px;  
 width: 230px;  
 background-color: var(--Card);  
 border-radius: 12px;  
 padding: 10px;  
 text-align: center;  
}  
  
.weapon-image {  
 object-fit: cover;  
 max-height: 95px;  
 width: 165px;  
 border-radius: 15px;  
}  
  
.weapon-name {  
 color: var(--Title-h1);  
 font-size: 1.2em;  
 margin: 10px 0;  
}  
  
.weapon-description {  
 color: var(--Title-h2);  
 font-size: 0.9em;  
 margin-bottom: 10px;  
}  
  
.weapon-price {  
 font-size: 1.1em;  
 margin-bottom: 10px;  
 color: var(--Button);  
}  
  
.add-to-cart {  
 background-color: transparent;  
 color: #ffffff;  
 cursor: pointer;  
}  
  
.down-cart {  
 width: 200px;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 gap: 10px;  
}  
  
.down-cart img {  
 width: 30px;  
}  
  
@media (max-width: 880px) {  
 .overview {  
 width: 740px;  
 }  
  
 .weapon-grid {  
 width: 735px;  
 grid-template-columns: repeat(3, 230px);  
 }  
}  
</style>

# **ГЛАВА 4. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БД**

## **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 и техническое задание можно выделить следующие сущности с их атрибутами (см. рисунок 32)

1. Сущность «Роль пользователя» с атрибутами «Код», «Название».
2. Сущность «Пользователь» с атрибутами «Имя», «Логин», «Пароль», «Почта», «Изображение»,».
3. Сущность «Заказ» с атрибутами «общая цена» и «статус оплаты».
4. Сущность «Корзина» с атрибутами «id продукта», «id заказа».
5. Сущность «Продукт» с атрибутами «Название», «Описание», «Цена», «Год».
6. Сущность «Изображение\_продукта» с атрибутом «Изображение».
7. Сущность «Тип категории» с атрибутом «Название».

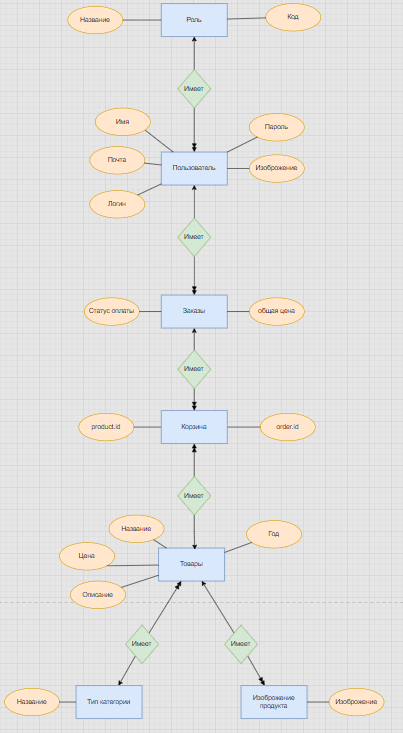


Рисунок 32 Концептуальная модель

## **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 (то есть факты, не выводимые из других хранимых фактов).

***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.

## **Логическая 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.

Используя нашу концептуальную модель данных и выполнив нормализацию данных до 3 нормальной формы включительно, построим логическую модель данных, в которой мы можем выделить новые атрибуты в таблице user, такие как логин, пароль и роль пользователя в системе, а также новые атрибуты, являющиеся первичными и внешними ключами, необходимые для построения связей.

Получившиеся логическая модель данных представлена на рисунке 33.

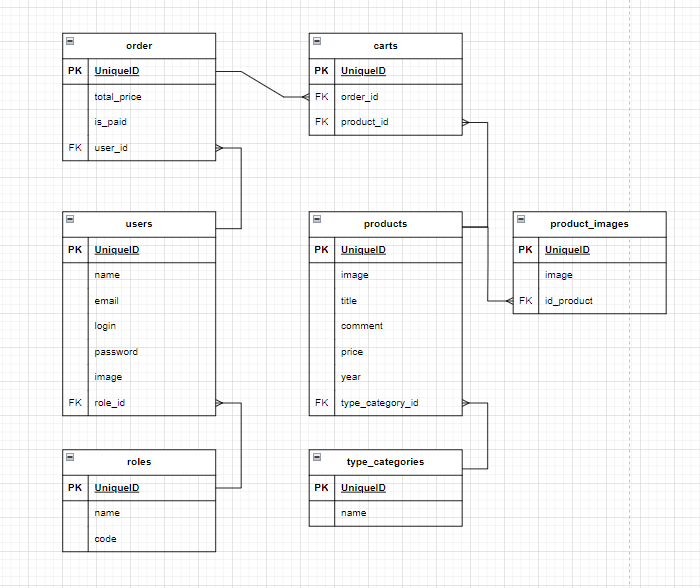


Рисунок 33 Логическая модель

## **Физическая модель данных**

Физическая модель данных – это модель данных, описанная с помощью средств конкретной системы управления базами данных (СУБД). Физическая модель данных строится на базе логической 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 34 представленная физическая модель данных для интернет-магазина.

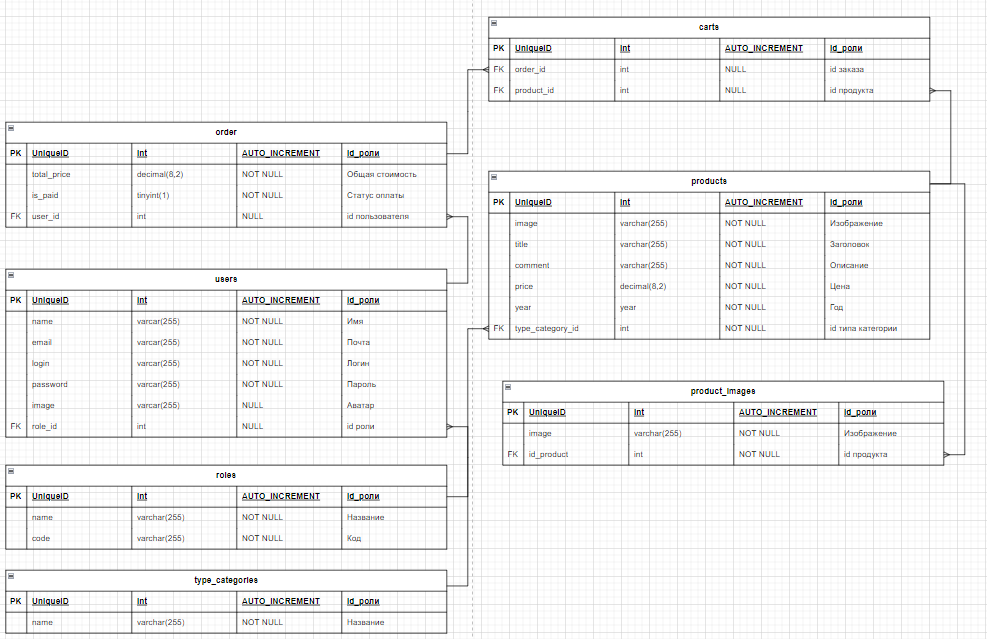


Рисунок 34 Физическая модель

# **ГЛАВА 5. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ API СЕРВИСА**

## **Описание проекта и задач**

**ОБЩЕЕ ОПИСАНИЕ ПРОЕКТА**

Все пользователи системы подразделяются на две группы:

1. Администраторы;
2. Пользователи;

Пользователи получают доступ к функциям ИС только после успешной авторизации, если пользователь не авторизирован, то ему доступны такие возможности, как просмотр товаров и фильтрация товара в каталоге.

Функционал администратора:

1. Вход в админ-панель;
2. Поиск товара в поисковой строке
3. Регистрация, редактирование, просмотр пользователей в системе;
4. Создание, редактирование, просмотр товаров;
5. Создание, редактирование, просмотр типов фильтрации;
6. Создание, редактирование, просмотр заказов и корзин;

Функционал зарегистрированного пользователя:

1. Просмотр и фильтрация товара
2. Добавление товара в корзину
3. Удаление товара из корзины
4. Поиск товара в поисковой строке
5. Просмотр профиля

Задача дипломного проекта – реализовать разработку backend используя фреймворк Laravel.

В примерах API запросов будут использоваться переменные {{API\_URL}}, которая обозначает адрес информационного продукта, и {{access\_token}}, которая обозначает присвоенный токен авторизации пользователя.

**ОБЩИЕ ТРЕБОВАНИЯ API**

API должно быть доступно с других доменов.

Все функции, кроме аутентификации, просмотра, поиска и фильтрации товара, доступны только авторизованным пользователям. Идентификацию пользователя организуется посредством Bearer Token.

При попытке доступа к защищенным авторизацией функциям системы во всех запросах необходимо возвращать ответ следующего вида:

**Status:** 401

**Accept:** application/json

**Body:** {  
 "message": "Unauthenticated."  
 }

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

{  
 "error": {  
 "message": <message>,  
 "errors": {  
 <key>: [<error message>]  
 }  
 }  
}

Вместо <message> необходимо указывать соответствующее значение, определенное в описании ответа на соответствующий запрос. В свойстве error.errors необходимо перечислить те свойства, которые не прошли валидацию, а в их значениях указать массив с ошибками валидации.

Например, если отправить пустой запрос на сервер, где проверяется следующая валидация:

* login – обязательно поле
* password – обязательное поле

то тело ответа будет следующим:

{

    "message": "validation.required (and 1 more error)",

    "errors": {

        "login": [

            "validation.required"

        ],

        "password": [

            "validation.required"

        ]

    }

}

Примите во внимание, что message могут быть определены иначе, если в запросе указано иное. В значениях свойств errors вы можете использовать любые сообщения об ошибках (если не указана конкретная ошибка), но они должны описывать возникшую проблему.

## **Специфические требования к API**

**Аутентификация**

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

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL**:{{API\_URL}}/login  **Method**: POST  **Headers**  - Accept: application/json  **Body**:  {  "login": "admin",  "password": "admin"  } | Успех  **Status**: 200  **Accept**: application/json  **Body**:  {  "data": {  "access\_token": <сгенерированный token>  "type\_token": "Bearer"  } } |
| Ошибки валидации полей  Формат ответа из общих требований |
| Неправильные логин или пароль  **Status**: 401  **Accept**: application/json  **Body**:  {  "error": {  "code": 401,  "message": "Пользователь не существует. Неверный логин или пароль"  } } |

**Выход**

Запрос предназначен для очистки значение токена пользователя.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL**:{{API\_URL}}/logout  **Method**: DELETE  **Auth Type:** Bearer Token | Успех  **Status**: 200  **Accept:** application/json  **Body**:  {  "data": {  "message": "Выход из учетной записи"  } } |

**Регистрация**

Запрос предназначен для регистрации пользователя

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL**:{{API\_URL}}/register  **Method**: POST  **Headers**  - Accept: application/json  **Body**:  {  "login": "admin",  "password": "admin"  "password\_confirmation": "admin"  "name": "admin"  "email": "admin@gmail.com"  } | Успех  **Status**: 200  **Accept**: application/json  **Body**:  {      "name": " admin ",      "login": " admin ",      "email": "admin@gmail.com ",      "updated\_at": "2024-05-16T14:01:59.000000Z",      "created\_at": "2024-05-16T14:01:59.000000Z",      "id": 3  } |
| Ошибки валидации полей  Формат ответа из общих требований |

## **Функционал пользователя**

**Просмотр всех товаров**

Запрос предназначен для просмотра всех товаров в любой из доступных категорий каталога на сайте. Параллельно этот же запрос производит фильтрацию по цене товара, если изменить значения, в поле фильтрации по цене, то отобразятся товары только те, которое входят в выбранный диапазон. При этом, если товара с такой ценой нет, запрос не выдаст результат. Так же фильтрует по типу категории: Платные, Бесплатные

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL**: {{API\_URL}}/api/product  **Method**: GET | Успех  **Status**: 200  **Accept**: application/json  **Body**:  {      "data": [          {              "id": 1,              "image": "01JVC3HNGBT2TVQPWTSXFW8715.jpg",              "title": "Triumph",              "comment": "Bike  ",              "price": "299.49",              "year": "2021",              "type\_category\_id": 1          },          {              "id": 2,              "image": "01JVC3JA2FBYHQ2VZWF7431EJC.jpg",              "title": "Yamaha",              "comment": "Sport bike",              "price": "1239.49",              "year": "2022",              "type\_category\_id": 2          }      ]  } |

**Поиск товара по названию**

Запрос выполняет функцию поисковой строки. Предназначен для поиска товаров по названию.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL**:{{API\_URL}}/search?search=Triumph  **Method**: GET  **Headers:**  - Accept: application/json | Успех  **Status**: 200  **Accept:** application/json  **Body**:  {      "query": "Triumph",      "products": [          {              "id": 1,              "image": "01JVC3HNGBT2TVQPWTSXFW8715.jpg",              "title": "Triumph",              "comment": "Bike  ",              "price": "299.49",              "year": "2021",              "type\_category\_id": 1,              "created\_at": **null**,              "updated\_at": "2025-05-16T08:32:05.000000Z"          }      ]  } |

**Просмотр корзины**

Запрос предназначен для просмотра корзины авторизированному пользователю.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL**:{{API\_URL}}/api/cart  **Method**: GET  **Headers:**  - Accept: application/json  **Auth Type:** Bearer Token | Успех  **Status**: 200  **Accept:** application/json  **Body**:  {      "data": [          {              "id": 2,              "order\_id": 1,              "product\_id": 1,              "order": {                  "id": 1,                  "total\_price": "799.48",                  "is\_paid": 0,                  "user\_id": 1,                  "user": {                      "name": "Kirill",                      "email": "admin@gmail.com",                      "login": "admin",                      "password": "$2y$12$v4cCvz7awTdX2RvAVWnOo..625DwNn7MmcmhwmLuP9RG0w7LiKuLa",                      "image": "",                      "role\_id": 1                  }              },              "product": {                  "id": 1,                  "image": "01JVC3HNGBT2TVQPWTSXFW8715.jpg",                  "title": "Triumph",                  "comment": "Bike  ",                  "price": "299.49",                  "year": "2021",                  "type\_category\_id": 1              }          }      ]  } |

**Добавление товара в корзину**

Запрос предназначен для добавления товара в корзину.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL**:{{API\_URL}}/api/cart/add-product/1  **Method**: POST  **Headers:**  - Accept: application/json  **Auth Type:** Bearer Token | Успех  **Status**: 201  **Accept:** application/json  **Body**:  {      "product": {          "id": 1,          "image": "01JVC3HNGBT2TVQPWTSXFW8715.jpg",          "title": "Triumph",          "comment": "Bike  ",          "price": "299.49",          "year": "2021",          "type\_category\_id": 1,          "created\_at": **null**,          "updated\_at": "2025-05-16T08:32:05.000000Z"      },      "cart\_item": {          "product\_id": "1",          "order\_id": 1,          "updated\_at": "2025-06-17T04:04:57.000000Z",          "created\_at": "2025-06-17T04:04:57.000000Z",          "id": 5      }  } |

**Удаление товара из корзины**

Запрос предназначен для удаление товара из корзины.

|  |  |
| --- | --- |
| **Request** | **Response** |
| **URL**:{{API\_URL}}/api/cart/delete-product/1  **Method**: DELETE  **Headers:**  - Accept: application/json  **Auth Type:** Bearer Token | Успех  **Status**: 200  **Accept:** application/json  **Body**:  {      "message": "Продукт удален из корзины"  } |
| Ошибка  **Status**: 404  **Accept:** application/json  **Body**:  {      "error": "Продукт не найден в корзине"  } |

## **Функционал Администратора**

Администратор имеет все возможности пользователя, которые описаны ранее.

**Авторизация в админ-панели**

У Администратора есть доступ к админ-панели, авторизация происходит по пути *local.host/admin/login (см. рисунок 35)*

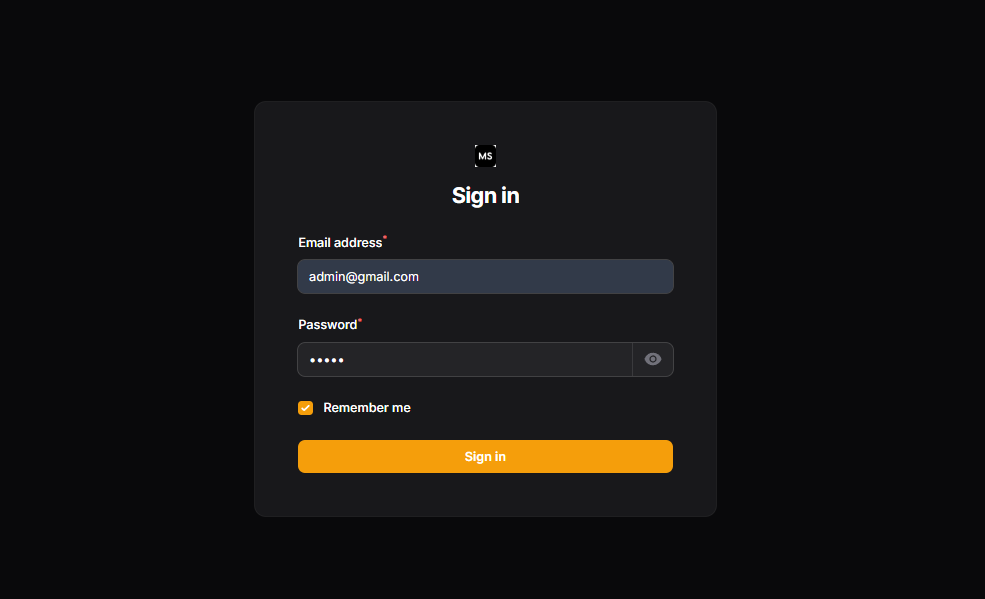


Рисунок 35 – Авторизация

**Управление админ-панелью**

На главной странице админ-панели предоставлена информация о зарегистрированных пользователях, заказах, доступных товаров (см. рисунок 36). Меню мини-профиля, которое предоставляет возможность смены темы на темную, светлую и системную, а также выход из админ-панели (см. рисунок 37). Меню навигации, которое предоставляет доступ к управлению корзиной, товарами, категориями, ролями, пользователями (см. рисунок 38).

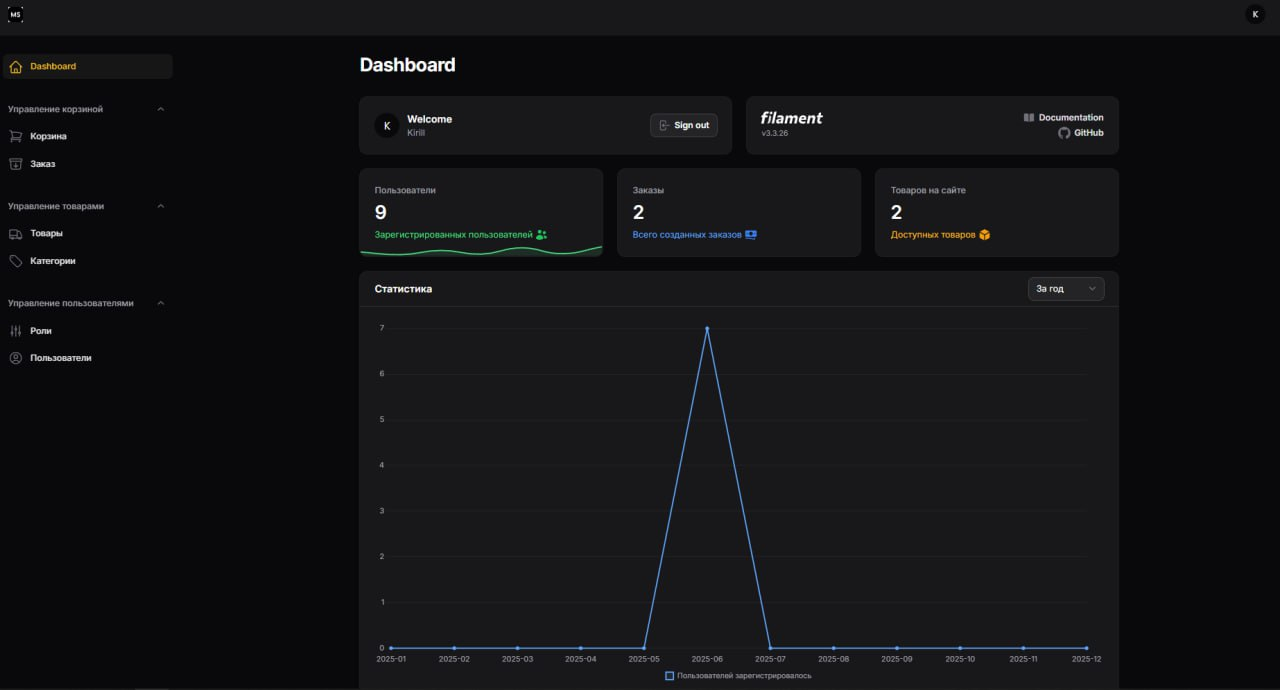


Рисунок 36 – Главное меню

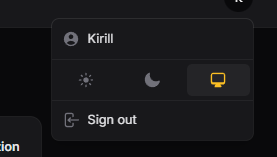


Рисунок 37 – Меню мини-профиля

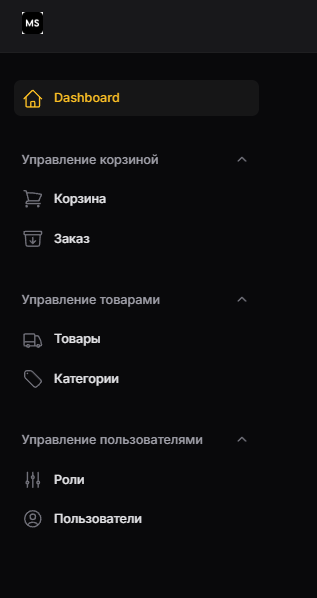


Рисунок 38 – Меню навигации

**Управление Корзинами**

На главной странице управление корзиной предоставлена информация о активных корзинах пользователей, которую можно по надобности очистить, либо редактировать. (см. рисунок 39).

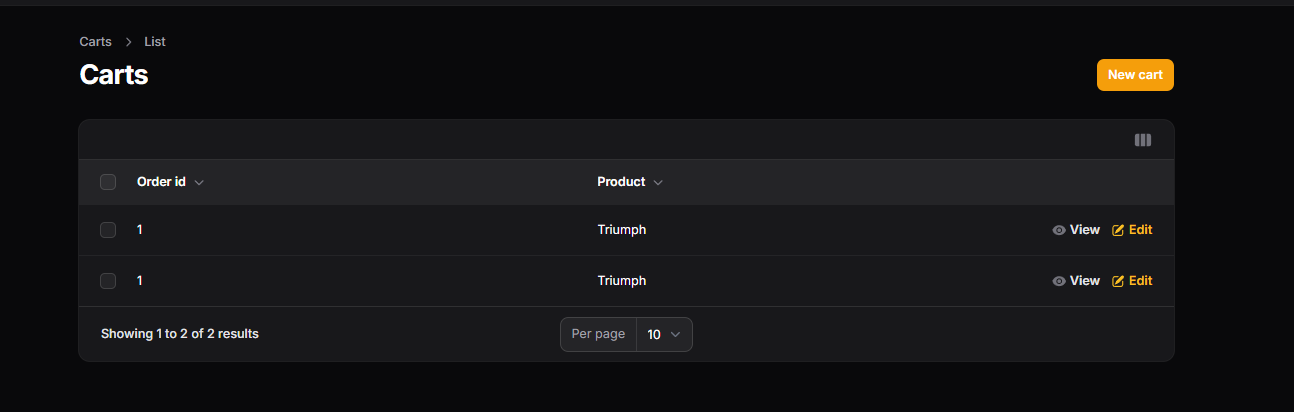


Рисунок 40 - Главная страница корзины

**Управление Заказами**

На главной странице управление заказами предоставлена информация о активных заказах пользователей, которую можно по надобности очистить, либо редактировать. (см. рисунок 41).

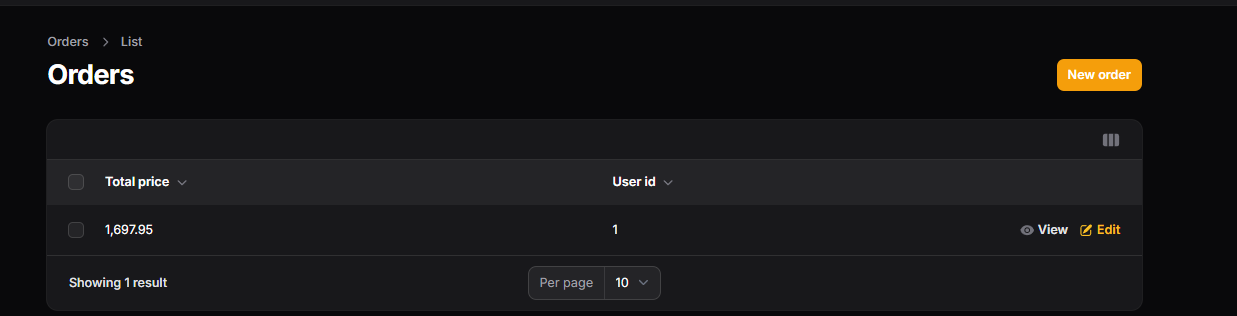


Рисунок 41 – Главная страница заказов

**Управление Товарами**

На главной странице управление товарами предоставлена информация о всех доступных товарах, которые можно по надобности удалить, создать, редактировать. (см. рисунок 42-44).

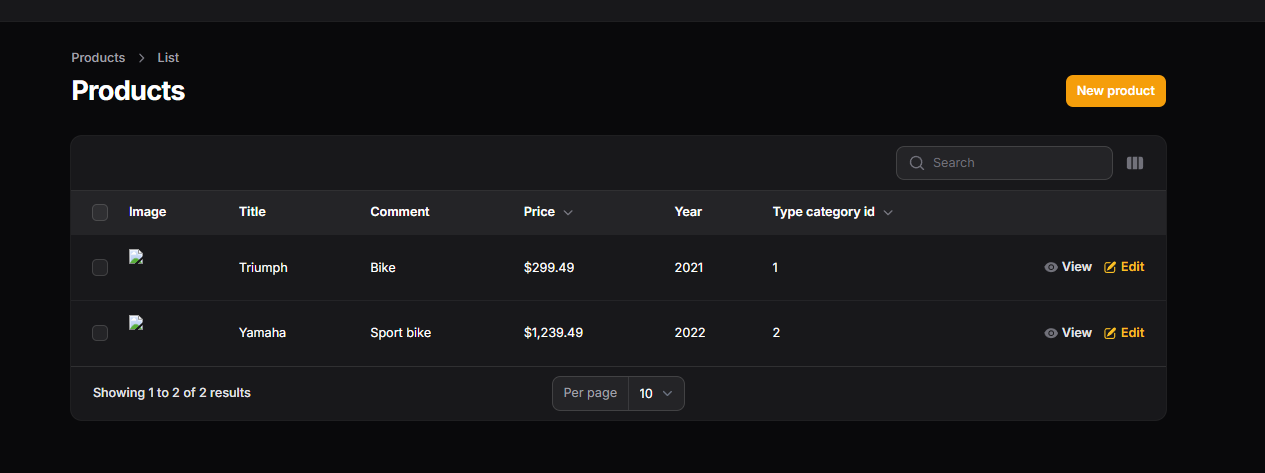


Рисунок 42 - Главное меню товаров

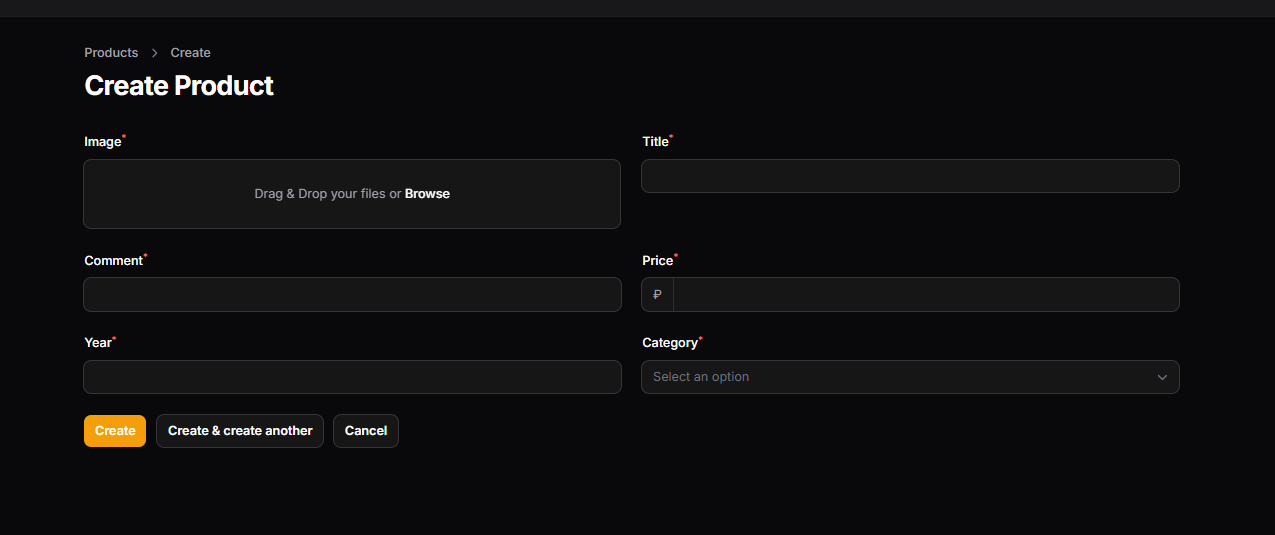
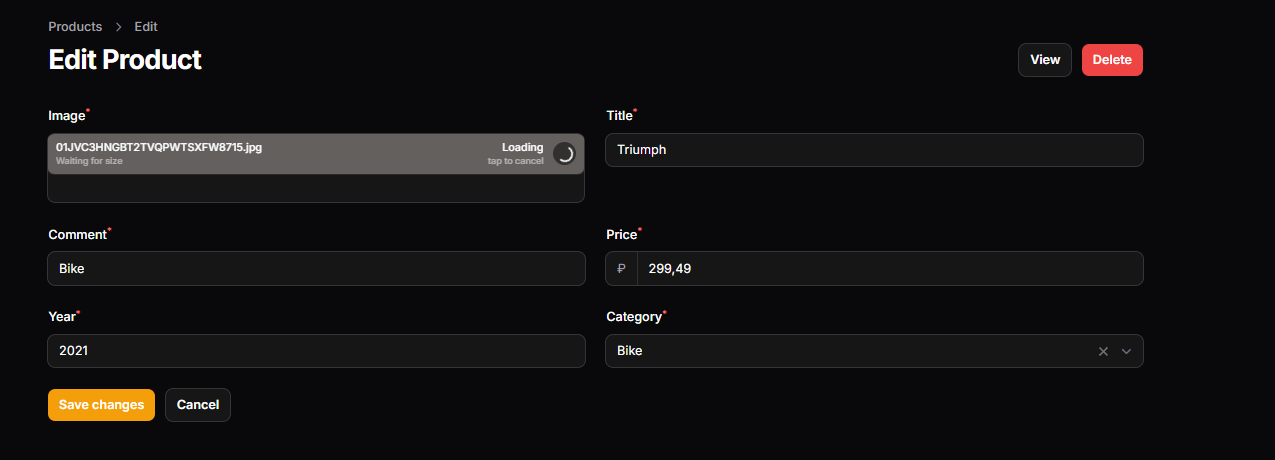


Рисунок 43- Меню создание товара

* Рисунок 44 - Меню редактирования товара*

**Управление Категориями**

На главной странице управление категориями предоставлена информация о активных категориях, которые можно по надобности удалить, создать, редактировать. (см. рисунок 45).

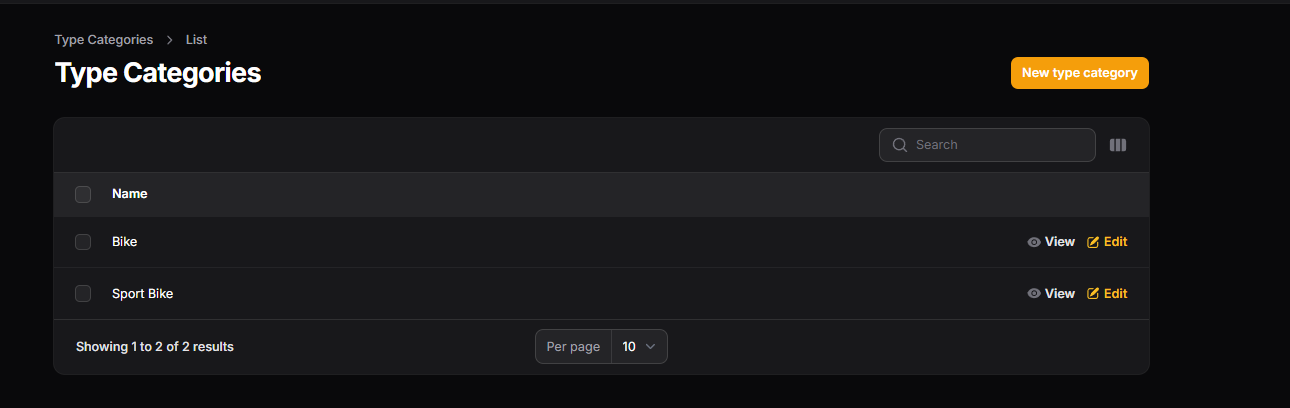


Рисунок 45 -Главное меню категорий

**Управление Ролями**

На главной странице управление ролями предоставлена информация о активных ролях, которые можно по надобности удалить, создать, редактировать. (см. рисунок 32).

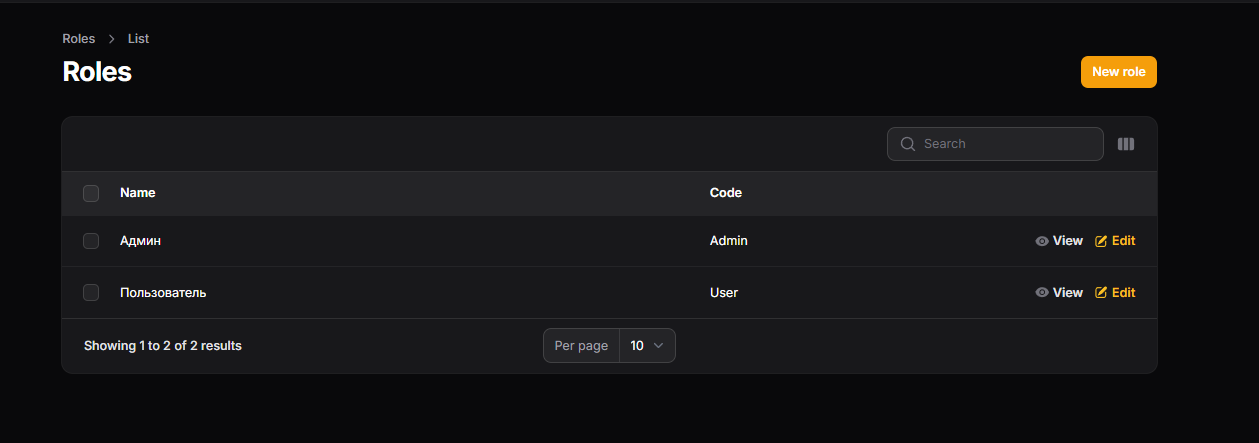


Рисунок 46 - Главное меню ролей

**Управление Пользователями**

На главной странице управление пользователями предоставлена информация о зарегистрированных пользователях, которых можно по надобности удалить, создать, редактировать. (см. рисунок 47).

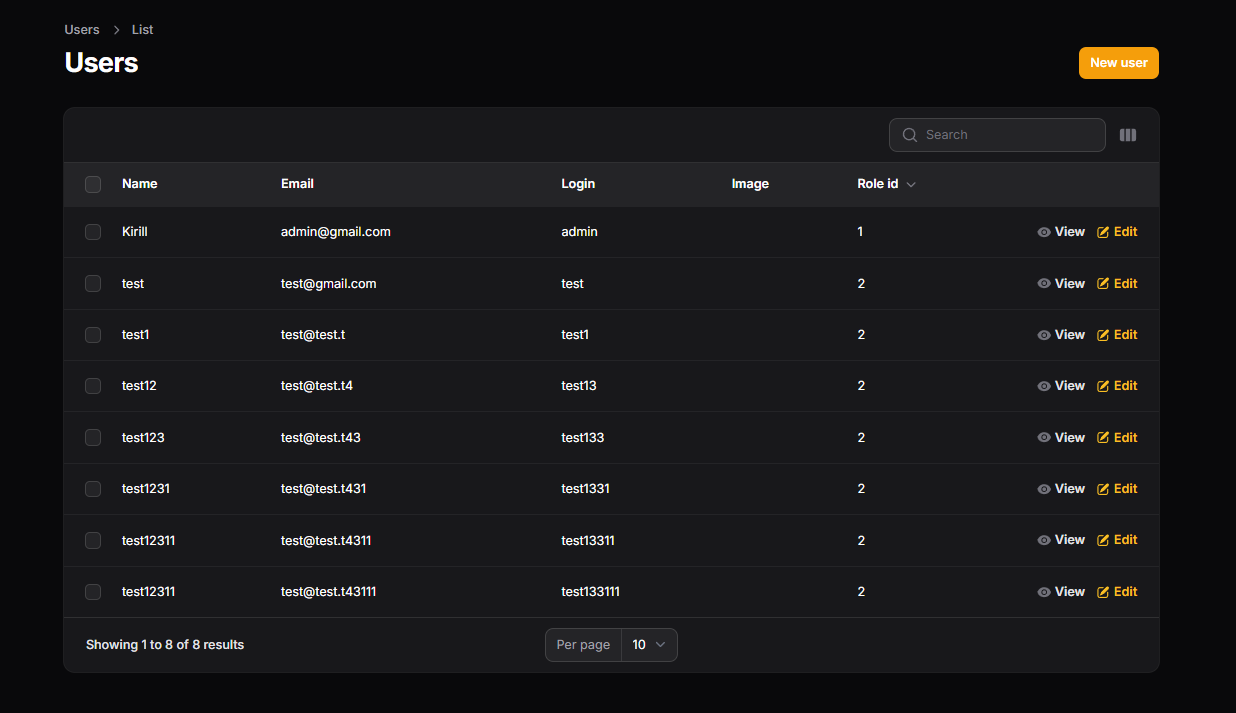


Рисунок 47 - Главное меню пользователей

# **ГЛАВА 6. ПРОЕКТИРОВАНИЕ И РАЗАРБОТКА API СЕРВИСА**

## **Установка и настройка фреймворка Laravel**

Для установки фреймворка в консоли сервера необходимо перейти в рабочую директорию сайта (используя команду cd) и выполнить команду обновления Composer’а, далее выполнить команду установку фреймворка Laravel.

composer create-project laravel/laravel motoshop “10.\*”.

В корневой директории сайта создаем файл .htaccess и прописываем правила перенаправления запросов в точку входа в приложение (в папку public) (см. Листинг 12).

Листинг 12 – Правила .htaccess

RewriteEngine on  
RewriteRule (.\*)? /public/$1

Редактируем файл конфигурации .env указав данные для подключения к БД MySQL (см. Листинг 13).

Листинг 13 – Файл конфигурации .env

DB\_CONNECTION=mysql  
DB\_HOST=127.0.0.1  
DB\_PORT=3306  
DB\_DATABASE=motoshop.ru  
DB\_USERNAME=root  
DB\_PASSWORD=

Редактируем файл конфигурации config\auth.php указав драйвер авторизации по API через систему токенов в разделе Authentication Guards (см. Листинг 14).

Листинг 14 - файл конфигурации config\auth.php.

'guards' => [  
 ...  
 'api' => [  
 'driver' => 'token',  
 'provider' => 'users',  
 'hash' => false,  
 ],  
],

Удаляем все стандартные миграции в папке database\migrations.

## **Создание миграций**

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.

Создаем миграции для таблиц roles, type\_categories, users, orders, products, carts 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\_orders\_table

php artisan make:migration create\_password\_reset\_tokens\_table

php artisan make:migration create\_password\_resets\_table

php artisan make:migration create\_type\_categories\_table

php artisan make:migration create\_products\_table

php artisan make:migration create\_carts\_table

Редактируем схему миграции таблицы roles.

Листинг 15 - схема миграции таблицы roles

public function up(): void  
{  
 Schema::create('roles', function (Blueprint $table) {  
 $table->id();  
 $table->string('name');  
 $table->string('code');  
 $table->timestamps();  
 });  
}

Редактируем схему миграции таблицы password\_reset\_tokens.

Листинг 16 - схема миграции таблицы password\_reset\_tokens.

public function up(): void  
{  
 Schema::create('password\_reset\_tokens', function (Blueprint $table) {  
 $table->string('email')->primary();  
 $table->string('token');  
 $table->timestamp('created\_at')->nullable();  
 });  
}

Редактируем схему миграции таблицы users.

Листинг 17 - схема миграции таблицы users.

public function up(): void  
{  
 Schema::create('users', function (Blueprint $table) {  
 $table->id();  
 $table->string('name');  
 $table->string('email')->unique();  
 $table->string('login');  
 $table->string('password');  
 $table->string('image')->nullable();  
 $table->unsignedBigInteger('role\_id')->default(2);  
  
 $table->foreign('role\_id')->references('id')->on('roles');  
  
 $table->rememberToken();  
 $table->timestamps();  
 });  
}

Редактируем схему миграции таблицы orders.

Листинг 18 - схема миграции таблицы orders.

public function up(): void  
{  
 Schema::create('orders', function (Blueprint $table) {  
 $table->id();  
 $table->decimal('total\_price', 8, 2);  
 $table->boolean('is\_paid')->default(false);  
 $table->unsignedBigInteger('user\_id')->nullable();  
  
 $table->foreign('user\_id')->references('id')->on('users');  
  
 $table->timestamps();  
 });  
}

Редактируем схему миграции таблицы type\_categories.

Листинг 19 - схема миграции таблицы type\_categories.

public function up(): void  
{  
 Schema::create('type\_categories', function (Blueprint $table) {  
 $table->id();  
 $table->string('name');  
 $table->timestamps();  
 });  
}

Редактируем схему миграции таблицы products.

Листинг 20 - схема миграции таблицы products.

public function up(): void  
{  
 Schema::create('products', function (Blueprint $table) {  
 $table->id();  
 $table->string('image');  
 $table->string('title');  
 $table->string('comment');  
 $table->decimal('price', 8, 2);  
 $table->year('year');  
 $table->unsignedBigInteger('type\_category\_id');  
  
 $table->foreign('type\_category\_id')->references('id')->on('type\_categories');  
  
 $table->timestamps();  
 });  
}

Редактируем схему миграции таблицы carts.

Листинг 21 - схема миграции таблицы carts.

public function up(): void  
{  
 Schema::create('carts', function (Blueprint $table) {  
 $table->id();  
 $table->unsignedBigInteger('order\_id')->nullable();  
 $table->unsignedBigInteger('product\_id')->nullable();;  
  
 $table->foreign('product\_id')->references('id')->on('products');  
 $table->foreign('order\_id')->references('id')->on('orders');  
  
 $table->timestamps();  
 });  
}

Редактируем схему миграции таблицы password\_resets.

Листинг 22 - схема миграции таблицы password\_resets.

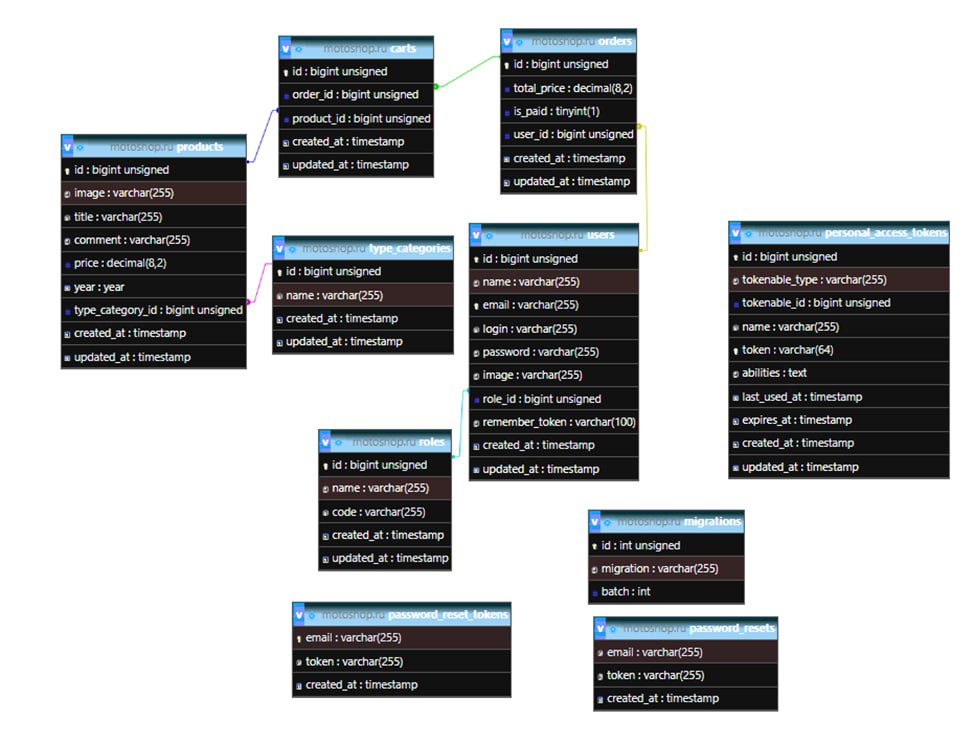
public function up(): void  
{  
 Schema::create('password\_resets', function (Blueprint $table) {  
 $table->string('email')->index();  
 $table->string('token');  
 $table->timestamp('created\_at')->nullable();  
 });  
}

Выполняем миграцию базы данных с помощью команды:

php artisan migrate

Диаграмма базы данных

На рисунке 36 представлена итоговая диаграмма базы данных для интернет-магазина.



Далее заполняем базу данных первоначальными данными.

## **Генерация классов моделей**

Laravel содержит ORM-библиотеку Eloquent, предоставляющую способ работы с базой данных, который часто удобнее обычного построителя запросов. При использовании Eloquent каждая таблица БД имеет соответствующую «Модель», которая используется для взаимодействия с этой таблицей. Помимо получения записей из таблицы БД, модели Eloquent также позволяют вставлять, обновлять и удалять записи из таблицы.

Модели расширяют класс Illuminate\Database\Eloquent\Model. Чтобы сгенерировать новую модель Eloquent, необходимо использовать Artisan-команду make:model. Эта команда поместит новый класс модели в каталог app/Models вашего приложения.

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

php artisan make:model Cart

php artisan make:model TypeCategory

php artisan make:model Order

php artisan make:model User

php artisan make:model Role

php artisan make:model Product

Далее отредактируем наши модели указав:

* Массово назначаемые атрибуты (переменная $fillable).
* Атрибуты, которые должны быть скрыты для массивов (переменная $hidden).
* Отношения.
* Дополнительные методы для получения и/или изменения данных в зависимости от технического задания.
* В конечном итоге наши модели будут имеет следующий вид:

Модель Cart (см. лист. 23 представляет сущность корзины в системе и служит связующим звеном между заказами и продуктами. В модели определены основные атрибуты, доступные для массового заполнения, включая идентификаторы самой корзины, заказа и продукта.

Модель реализует методы для установления связей с другими таблицами: каждая корзина принадлежит одному заказу и связана с одним продуктом. Эти связи позволяют эффективно извлекать информацию о содержимом заказа, а также о продуктах, включённых в него.

Листинг 23 – Модель Cart.

class Cart extends Model  
{  
 *//Атрибуты, которые можно массово назначать.* protected $fillable = [  
 'id',  
 'order\_id',  
 'product\_id',  
 ];  
  
 *//Получить заказ, связанный с корзиной.* public function order(): BelongsTo  
 {  
 return $this->belongsTo(Order::class);  
 }  
  
 *//Получить продукт, связанный с корзиной.* public function product(): BelongsTo  
 {  
 return $this->belongsTo(Product::class);  
 }  
  
  
}

Модель Order (см. лист. 24) представляет сущность заказа в системе и определяет его основные атрибуты и связи с другими таблицами. В модели указаны поля, доступные для массового заполнения, такие как идентификатор заказа, общая стоимость, идентификатор пользователя и статус оплаты.

Модель содержит методы для установления связей с другими моделями: заказ принадлежит одному пользователю, связан с несколькими корзинами, а также имеет отношение ко многим продуктам через промежуточную таблицу корзин, включая дополнительное поле is\_paid. Эти отношения позволяют удобно извлекать данные о пользователе, товарах и корзинах, связанных с конкретным заказом.

Листинг 24 – Модель Order.

class Order extends Model  
{  
 *//Атрибуты, которые можно массово назначать.* protected $fillable = [  
 'id',  
 'total\_price',  
 'user\_id',  
 'is\_paid',  
 ];  
  
 *//Получить пользователя, связанного с заказом.* public function user(): BelongsTo  
 {  
 return $this->belongsTo(User::class);  
 }  
  
 *//Получить корзины, связанные с заказом.* public function carts(): HasMany  
 {  
 return $this->hasMany(Cart::class);  
 }  
  
 *//Получить продукты, связанные с заказом через корзины.* public function products():BelongsToMany  
 {  
 return $this->belongsToMany(Product::class, Cart::class)->withPivot('is\_paid');  
 }  
}

Модель Product (см. лист. 25) представляет сущность продукта в системе и определяет его ключевые атрибуты и связи с другими таблицами. В перечень полей, доступных для массового заполнения, входят идентификатор продукта, изображение, название, описание, цена, год выпуска и идентификатор категории.

Модель реализует несколько связей: каждый продукт принадлежит одной категории, может быть включён в несколько корзин, а также участвовать в различных заказах через промежуточную таблицу. Для связи с категорией используется внешний ключ type\_category\_id, а также предусмотрен дополнительный метод, реализующий аналогичную связь. Такие связи обеспечивают удобный доступ к информации о принадлежности продукта к категории, а также о его участии в заказах.

Кроме того, в модели реализован метод getProductByPrice, позволяющий получить все продукты в заданном диапазоне цен, что расширяет функциональность модели при фильтрации данных.

Листинг 25 – Модель Product.

class Product extends Model  
{  
 *//Атрибуты, которые можно массово назначать.* protected $fillable = [  
 'id',  
 'image',  
 'title',  
 'comment',  
 'price',  
 'year',  
 'type\_category\_id',  
 ];  
  
 *//Получить категорию продукта.* public function category():BelongsTo  
 {  
 return $this->belongsTo(TypeCategory::class, 'type\_category\_id');  
 }  
  
 *//Получить корзины, содержащие данный продукт.* public function cart():HasMany  
 {  
 return $this->hasMany(Cart::class);  
 }  
  
 *//Получить заказы, в которых содержится данный продукт.* public function orders():BelongsToMany  
 {  
 return $this->belongsToMany(Order::class);  
 }  
  
 *//Получить категорию продукта.* public function TypeCategory():BelongsTo  
 {  
 return $this->belongsTo(TypeCategory::class);  
 }  
  
 public function getProductByPrice($priceStar, $priceEnd)  
 {  
 return $this->whereBetween('price', [$priceStar, $priceEnd])->get();  
 }  
}

Модель Role (см. лист. 26) представляет сущность пользовательской роли в системе и определяет основные атрибуты, доступные для массового заполнения, такие как название роли и её кодовое обозначение. Эта модель может использоваться для разграничения прав доступа и организации ролевой модели в приложении.

Связи с другими моделями в данном листинге не определены, однако модель может быть расширена для установления отношений с пользователями или другими сущностями системы, если это потребуется в рамках логики авторизации и аутентификации.

Листинг 26 – Модель Role.

class Role extends Model  
{  
 *//Атрибуты, которые можно массово назначать.* protected $fillable = [  
 'name',  
 'code',  
 ];  
}

Модель TypeCategory (см. лист. 27) представляет сущность типа категории товара в системе и определяет её основные характеристики. К полям, доступным для массового заполнения, относятся идентификатор и наименование категории.

В модели реализована связь с продуктом через метод Product, указывающий на принадлежность к модели Product. Однако с точки зрения бизнес-логики, один тип категории, как правило, должен быть связан с несколькими продуктами, поэтому целесообразнее было бы использовать связь hasMany вместо belongsTo. Теклая реализация может ограничивать корректное отражение отношений между категориями и товарами.

Листинг 27 – Модель TypeCategory.

class TypeCategory extends Model  
{  
 *//Атрибуты, которые можно массово назначать.* protected $fillable = [  
 'id',  
 'name',  
 ];  
  
 *//Получить продукт, связанный с этим типом категории.* public function Product():BelongsTo  
 {  
 return $this->belongsTo(Product::class);  
 }  
}

Модель User (см. лист. 28) представляет сущность пользователя в системе и включает в себя основные поля, такие как имя, электронная почта, логин, пароль, изображение профиля и идентификатор роли. Эти поля доступны для массового заполнения. Кроме того, модель скрывает чувствительные данные, такие как пароль и токен запоминания, от вывода, а также использует преобразование даты подтверждения электронной почты к нативному типу datetime.

Модель реализует связи с другими сущностями: каждый пользователь принадлежит одной роли, может иметь несколько заказов, а также взаимодействовать с продуктами через механизм избранного (многие-ко-многим). Это обеспечивает гибкое управление данными, связанными с пользователем, включая его заказы, привилегии и предпочтения.

Дополнительно в модели определены обработчики событий создания и обновления, обеспечивающие автоматическое хеширование пароля перед сохранением в базу данных. Это повышает уровень безопасности системы.

Листинг 28 – Модель User.

class User extends *Authenticatable*{  
 use HasApiTokens, HasFactory, Notifiable;  
  
 *//Атрибуты, которые можно массово назначать.* protected $table = 'users';  
 protected $fillable = [  
 'name',  
 'email',  
 'login',  
 'password',  
 'image',  
 'role\_id',  
 ];  
 *//Атрибуты, которые скрыты от вывода.* protected $hidden = [  
 'password',  
 'remember\_token',  
 ];  
 *//Преобразование атрибутов в нативные типы.* protected $casts = [  
 'email\_verified\_at' => 'datetime',  
 ];  
  
 *// Получить роль пользователя.* public function role():BelongsTo  
 {  
 return $this->belongsTo(Role::class);  
 }  
  
 *//Получить заказы пользователя.* public function orders(): HasMany  
 {  
 return $this->hasMany(Order::class);  
 }  
  
 *//Получить избранные продукты пользователя.* public function favorites(): BelongsToMany  
 {  
 return $this->belongsToMany(Product::class, 'favorites');  
 }  
  
 *//Обработчик события создания модели.* public static function boot(): void  
 {  
 parent::*boot*();  
  
 static::*creating*(function ($user) {  
 $user->password = Hash::make($user->password);  
 });  
  
 static::*updating*(function ($user) {  
 if ($user->isDirty('password')) {  
 $user->password = Hash::make($user->password);  
 }  
 });  
 }  
}

Ниже будут описаны Request запросы для различных запросов по техническому заданию:

Класс валидации запроса формы OrderRequest – добавление заказа. Заказ может создать только авторизированный пользователь.

Модель OrderRequest (см. лист. 29) представляет собой класс запроса, предназначенный для обработки и валидации входящих данных при создании заказа. Этот класс наследуется от базового класса FormRequest и используется для обеспечения целостности и корректности передаваемых данных.

Метод authorize возвращает значение true, что позволяет любому пользователю отправлять данный запрос. Метод rules содержит набор правил валидации, определяющий обязательность полей total\_price и is\_paid при передаче данных. Это гарантирует, что необходимые параметры заказа будут заполнены корректно до попадания в контроллер.

Класс OrderRequest способствует разделению логики валидации и обработки данных, улучшая читаемость и поддержку кода.

Листинг 29 – Класс валидации OrderRequest.

class OrderRequest extends FormRequest  
{  
 */\*\*  
 \* Determine if the user is authorized to make this request.  
 \*/* public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules()  
 {  
 return [  
 'total\_price' => ['required'],  
 'is\_paid' => ['required'],  
 ];  
 }  
  
}

Модель CartRequest (см. лист. 30) представляет собой класс запроса, предназначенный для валидации данных при добавлении товара в корзину. Класс наследуется от базового класса FormRequest и используется для предварительной проверки данных, поступающих от пользователя.

Метод authorize возвращает значение false, что на текущем этапе ограничивает возможность выполнения запроса. Согласно бизнес-логике, добавление товара в корзину должно быть доступно только авторизованным пользователям. Для реализации этого поведения требуется изменить значение на true и при необходимости добавить дополнительную проверку авторизации.

Метод rules пока не содержит правил валидации, однако может быть дополнен в будущем для проверки, например, идентификатора товара или количества.

Модель CartRequest способствует структурированию логики валидации и позволяет централизованно управлять правилами при добавлении товаров в корзину.

Листинг 30 – Класс валидации CartRequest.

class CartRequest extends FormRequest  
{  
 public function authorize(): bool  
 {  
 return false;  
 }  
 public function rules(): array  
 {  
 return [  
 *//* ];  
 }  
}

Модель ProductRequest (см. лист. 31) представляет собой класс запроса, предназначенный для валидации данных при создании или редактировании продукта. Класс наследуется от базового FormRequest и используется для обеспечения корректности и полноты передаваемой информации.

Метод authorize возвращает значение true, что позволяет выполнять данный запрос без дополнительных ограничений по правам доступа. Метод rules определяет набор обязательных полей: наименование (title), описание (comment), цена (price), а также год выпуска (year). Поле price дополнительно проверяется на соответствие числовому формату, что исключает возможность передачи некорректных данных.

Класс ProductRequest обеспечивает надёжную фильтрацию входящих данных и способствует повышению безопасности и стабильности системы при работе с сущностями продуктов.

Листинг 31 – Класс валидации ProductRequest.

class ProductRequest extends FormRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules()  
 {  
 return [  
 'title' => ['required'],  
 'comment' => ['required'],  
 'price' => ['required', 'numeric'],  
 'year' => ['required'],  
 ];  
 }  
  
}

Класс валидации RoleRequest (см. лист. 32) предназначен для проверки корректности данных при создании или обновлении роли пользователя. Класс наследуется от базового FormRequest и обеспечивает централизованную валидацию входящих параметров.

Метод authorize возвращает true, позволяя выполнять запрос без ограничений по правам доступа. Метод rules устанавливает обязательность полей name и code, что гарантирует наличие названия роли и её кодового обозначения при передаче данных.

Данный класс валидации способствует поддержанию целостности данных и повышает надёжность процесса управления ролями в системе.

Листинг 32 – Класс валидации RoleRequest.

class RoleRequest extends FormRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'name' => ['required'],  
 'code' => ['required'],  
 ];  
 }  
}

Класс валидации TypeCategoryRequest (см. лист. 33) предназначен для проверки данных при создании или обновлении типа категории товара. Класс наследуется от FormRequest и обеспечивает проверку входящих данных на соответствие заданным правилам.

Метод authorize возвращает true, что позволяет выполнять запрос без ограничений по правам доступа. В методе rules определено обязательное поле name, гарантирующее, что название типа категории будет обязательно заполнено.

Данный класс валидации способствует поддержанию корректности данных и упрощает управление типами категорий в системе.

Листинг 33 – Класс валидации TypeCategoryRequest.

class TypeCategoryRequest extends FormRequest  
{  
 public function authorize(): bool  
 {  
 return true;  
 }  
 public function rules(): array  
 {  
 return [  
 'name' => ['required'],  
 ];  
 }  
}

## **Генерация классов контроллера**

Вместо того чтобы определять всю логику обработки запросов как замыкания в файлах маршрутов, вы можете организовать это поведение с помощью классов «контроллеров». Контроллеры могут сгруппировать связанную логику обработки запросов в один класс.

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

php artisan make:controller AuthUserController

php artisan make:controller CartController

php artisan make:controller Controller

php artisan make:controller OrderController

php artisan make:controller PersonalAccessTokenController

php artisan make:controller ProductController

php artisan make:controller RoleController

php artisan make:controller SearchController

php artisan make:controller TypeCategoryController

Далее отредактируем наши котроллеры написав необходимые методы:

Контроллер AuthUserController (см. лист. 34) отвечает за регистрацию, аутентификацию и выход пользователя из системы. В контроллере реализованы методы для обработки соответствующих HTTP-запросов.

Метод store выполняет валидацию входящих данных при регистрации пользователя, включая проверку уникальности логина и электронной почты, а также требований к паролю. После успешной проверки создаётся новый пользователь с хешированным паролем, после чего происходит его автоматическая авторизация и возврат успешного ответа с данными пользователя.

Метод login обрабатывает попытку входа, выполняя валидацию логина и пароля, затем пытается аутентифицировать пользователя. При успешном входе создаётся и возвращается токен доступа, используемый для последующей аутентификации в системе. В случае ошибки возвращается сообщение об ошибке.

Метод logout обеспечивает выход пользователя из системы путём удаления текущего токена доступа и возвращает подтверждение успешного выхода.

Контроллер способствует реализации безопасного и удобного механизма управления сессиями пользователей.

Листинг 34 – Контроллер AuthUserController.

class AuthUserController extends Controller  
{  
 public function store(Request $request)  
 {  
 *// Валидация данных запроса* $validated = $request->validate([  
 'name' => ['required', 'string', 'max:255'],  
 'login' => ['required', 'string', 'max:255', 'unique:'.User::class],  
 'email' => ['required', 'string', 'email', 'max:255', 'unique:'.User::class],  
 'password' => ['required', 'string', 'confirmed', Password::*defaults*()],  
 ]);  
  
 *// Создание нового пользователя* $user = User::*create*([  
 'name' => $validated['name'],  
 'login' => $validated['login'],  
 'email' => $validated['email'],  
 'password' => bcrypt($validated['password']),  
 ]);  
  
 *// Авторизация пользователя* Auth::login($user);  
  
 return response()->json([  
 'message' => 'Пользователь успешно зарегистрирован',  
 'user' => $user->only(['id', 'name', 'email', 'login'])  
 ], 201);  
 }  
  
 public function login(Request $request)  
 {  
 *// Валидация данных запроса* $request->validate([  
 'login' => ['required', 'string', 'max:255'],  
 'password' => ['required', 'string'],  
 ]);  
  
 *// Получение учетных данных пользователя* $credentials = $request->only('login', 'password');  
  
 *// Попытка аутентификации пользователя* if (Auth::attempt($credentials)) {  
 *// Создание токена для аутентифицированного пользователя* $token = $request->user()->createToken('authToken')->plainTextToken;  
 *// Возвращение ответа с токеном доступа* return response()->json([  
 'access\_token' => $token,  
 'token\_type' => 'Bearer',  
 ]);  
 }  
  
 *// Возвращение ошибки при неудачной аутентификации* return response()->json(['error' => 'Пользователь не существует. Неверный логин или пароль'], 401);  
 }  
  
 public function logout(Request $request)  
 {  
 *// Удаление текущего токена пользователя* $request->user()->currentAccessToken()->delete();  
  
 return response()->json([  
 'message' => 'Успешный выход из системы'  
 ]);  
 }  
}

Контроллер CartController (см. лист. 35) управляет операциями с элементами корзины пользователя и обеспечивает полный CRUD-функционал.

Метод index возвращает коллекцию всех элементов корзины в формате ресурса CartResource.

Метод store создаёт новый элемент корзины на основе валидированных данных из CartRequest.

Методы show, update и destroy предоставляют возможности просмотра, обновления и удаления конкретного элемента корзины соответственно.

Метод addItem добавляет товар в корзину текущего аутентифицированного пользователя. Если у пользователя отсутствует активный неоплаченный заказ, создаётся новый с общей стоимостью, равной цене добавляемого продукта. Если заказ уже существует, общая стоимость обновляется. Затем в заказ добавляется новый элемент корзины с указанным продуктом, и возвращается ответ с данными добавленного товара.

Метод removeItem удаляет товар из корзины активного заказа пользователя. При отсутствии активного заказа или если товар не найден, возвращаются соответствующие ошибки. Если после удаления товара корзина становится пустой, удаляется и сам заказ. В конце возвращается подтверждение успешного удаления.

Контроллер обеспечивает удобное и безопасное управление корзиной пользователя, учитывая состояние заказов и связи с продуктами.

Листинг 35 – Контроллер CartController.

class CartController extends Controller  
{  
 public function index()  
 {  
 *//Возвращает коллекцию всех элементов корзины.* return CartResource::*collection*(Cart::*all*());  
 }  
  
 public function store(CartRequest $request)  
 {  
 *//Создает новый элемент корзины.* return Cart::*create*($request->validated());  
 }  
  
 public function show(Cart $cart)  
 {  
 *//Возвращает информацию о конкретном элементе корзины.* return $cart;  
 }  
  
 public function update(CartRequest $request, Cart $cart)  
 {  
 *//Обновляет информацию о конкретном элементе корзины.* $cart->update($request->validated());  
  
 return $cart;  
 }  
  
 public function destroy(Cart $cart)  
 {  
 *//Удаляет элемент корзины.* $cart->delete();  
  
 return response()->json();  
 }  
  
 public function addItem(Request $request, $id)  
 {  
 *// Получаем информацию о товаре* $product = Product::*query*()->findOrFail($id);  
  
 $user = Auth::user();  
  
 *// Ищем активный заказ пользователя* $order = $user->orders()->where('is\_paid', false)->first();  
  
 *// Если заказ не существует, создаем новый заказ* if (!$order) {  
 $order = $user->orders()->create([  
 'is\_paid' => false,  
 'total\_price' => $product->price,  
 ]);  
 } else {  
 *// Если заказ существует, обновляем его общую сумму* $order->update([  
 'total\_price' => $order->total\_price + $product->price,  
 ]);  
 }  
  
 *// Используем create для создания нового элемента корзины* $cartItem = $order->carts()->create([  
 'product\_id' => $id,  
 ]);  
  
 *// Возвращаем ответ в виде JSON с кодом состояния 201* return response()->json([  
 'product' => $product,  
 'cart\_item' => $cartItem,  
 ], 201);  
 }  
  
 public function removeItem(Request $request, $productId)  
 {  
 *// Получаем текущего пользователя* $user = Auth::user();  
  
 *// Ищем текущий неоплаченный заказ пользователя* $order = $user->orders()->where('is\_paid', false)->first();  
 if (!$order) {  
 return response()->json(['error' => 'Нет активных заказов'], 404);  
 }  
  
 *// Ищем товар в корзине пользователя* $cartItem = $order->carts()->where('id', $productId)->first();  
 if (!$cartItem) {  
 return response()->json(['error' => 'Продукт не найден в корзине'], 404);  
 }  
  
 *// Удаляем товар из корзины* $cartItem->delete();  
  
 *// Проверяем, остались ли еще товары в заказе* if ($order->carts->isEmpty()) {  
 *// Если товаров не осталось, удаляем заказ* $order->delete();  
 }  
  
 return response()->json(['message' => 'Продукт удален из корзины'], 200);  
 }  
}

Контроллер Controller (см. лист. 36) является базовым контроллером приложения, который наследуется от BaseController. В нем используются трейты AuthorizesRequests и ValidatesRequests, обеспечивающие функциональность авторизации и валидации запросов соответственно.

Данный контроллер служит основой для всех остальных контроллеров в приложении, предоставляя им базовый набор возможностей и обеспечивая единообразие в обработке запросов.

Листинг 36 – Контроллер Controller.

class Controller extends *BaseController*{  
 use AuthorizesRequests, ValidatesRequests;  
}

Контроллер GetController (см. лист. 36) представляет собой однопроцессный контроллер с методом \_\_invoke, позволяющим использовать его как вызываемый объект. В текущей реализации метод \_\_invoke не содержит логики и обозначен как задача для дальнейшей реализации.

Данный контроллер может применяться для обработки простых маршрутов, где требуется единственная точка входа.

Листинг 36 – Контроллер GetController.

class GetController extends Controller  
{  
 public function \_\_invoke()  
 {  
 *// TODO: Implement \_\_invoke() method.* }  
}

Контроллер OrderController (см. лист. 37) отвечает за управление заказами в системе и реализует полный набор CRUD-операций.

Метод index возвращает коллекцию всех заказов в виде ресурса OrderResource.

Метод store создаёт новый заказ на основе валидированных данных из OrderRequest.

Методы show, update и destroy обеспечивают просмотр, обновление и удаление конкретного заказа соответственно.

Метод payOrder реализует логику оплаты заказа. Он находит заказ по идентификатору, проверяет, был ли он уже оплачен, и в случае успешной оплаты обновляет статус заказа. Если заказ уже оплачен, возвращается ошибка с соответствующим сообщением.

Данный контроллер обеспечивает удобное и безопасное управление жизненным циклом заказов.

Листинг 37 – Контроллер OrderController.

class OrderController extends Controller  
{  
 public function index()  
 {  
 *//Возвращает коллекцию всех заказов.* return OrderResource::*collection*(Order::*all*());  
 }  
  
 public function store(OrderRequest $request)  
 {  
 *//Создает новый заказ.* return Order::*create*($request->validated());  
 }  
  
 public function show(Order $order)  
 {  
 *//Возвращает информацию о конкретном заказе.* return $order;  
 }  
  
 public function update(OrderRequest $request, Order $order)  
 {  
 *//Обновляет информацию о конкретном заказе* $order->update($request->validated());  
  
 return $order;  
 }  
  
 public function destroy(Order $order)  
 {  
 *//Удаляет заказ.* $order->delete();  
  
 return response()->json();  
 }  
  
 public function payOrder(Request $request, $orderId)  
 {  
 *// Находим заказ* $order = Order::*findOrFail*($orderId);  
  
 *// Проверяем, оплачен ли уже заказ* if ($order->is\_paid) {  
 return response()->json(['error' => 'Заказ уже оплачен'], 400);  
 }  
  
 *// Проводим логику оплаты заказа (ваша логика здесь)  
  
 // После успешной оплаты обновляем статус заказа на оплаченный* $order->update([  
 'is\_paid' => true,  
 ]);  
  
 return response()->json(['message' => 'Заказ успешно оплачен'], 200);  
 }  
}

Контроллер PersonalAccessTokenController (см. лист. 38) отвечает за управление персональными токенами доступа пользователей для API.

Метод store выполняет валидацию данных запроса, проверяет существование пользователя по логину и корректность пароля. В случае успешной проверки создаёт новый персональный токен доступа и возвращает его в ответе.

Метод destroy удаляет указанный персональный токен пользователя по его идентификатору, обеспечивая контроль над активными токенами и безопасность доступа.

Данный контроллер способствует управлению механизмом аутентификации через персональные токены в системе.

Листинг 38 – Контроллер PersonalAccessTokenController.

class ProductController extends Controller  
{  
 *//Возвращает коллекцию всех товаров.* public function index(Request $request)  
 {  
 if ($request->filled(['priceStart', 'priceEnd'])) {  
 return ProductResource::*collection*(  
 (new Product)->getProductByPrice($request->priceStart, $request->priceEnd)  
 );  
 }  
  
 return ProductResource::*collection*(Product::*all*());  
 }  
  
 *//Создает новый товар.* public function store(ProductRequest $request)  
 {  
 return Product::*create*($request->validated());  
 }  
  
 *//Возвращает информацию о конкретном товаре.* public function show(Product $product)  
 {  
 return $product;  
 }  
  
 *//Обновляет информацию о конкретном товаре.* public function update(ProductRequest $request, Product $product)  
 {  
 $product->update($request->validated());  
  
 return $product;  
 }  
  
 *//Удаляет товар.* public function destroy(Product $product)  
 {  
 $product->delete();  
  
 return response()->json();  
 }  
  
 *//Фильтрует товары по категории.* public function filter(Request $request, $type\_category\_id)  
 {  
 *// Поиск товара по заданному идентификатору* $product = Product::*findOrFail*($type\_category\_id);  
  
 *// Получение категории товара* $category = $product->category()->where('id', $type\_category\_id)->first();  
  
 *// Проверка наличия категории* if (!$category) {  
 return response()->json(['error' => 'Нет таких товаров'], 404);  
 }  
  
 *// Фильтрация товаров по категории* $filteredProducts = Product::*where*('type\_category\_id', $type\_category\_id)->get();  
  
 *// Возвращение коллекции ресурсов отфильтрованных товаров* return ProductResource::*collection*($filteredProducts);  
 }  
}

Контроллер RoleController (см. лист. 39) реализует управление ролями пользователей в системе с полным набором CRUD-операций.

Метод index возвращает коллекцию всех ролей в формате ресурса RoleResource.

Метод store создаёт новую роль на основе валидированных данных из RoleRequest.

Методы show, update и destroy предоставляют возможности просмотра, обновления и удаления конкретной роли соответственно.

Данный контроллер обеспечивает удобное управление ролями и их атрибутами, поддерживая целостность и актуальность данных.

Листинг 39 – Контроллер RoleController.

class RoleController extends Controller  
{  
 *//Возвращает коллекцию всех ролей.* public function index()  
 {  
 return RoleResource::*collection*(Role::*all*());  
 }  
  
 *//Создает новую роль.* public function store(RoleRequest $request)  
 {  
 return Role::*create*($request->validated());  
 }  
  
 *//Возвращает информацию о конкретной роли.* public function show(Role $role)  
 {  
 return $role;  
 }  
  
 *//Обновляет информацию о конкретной роли.* public function update(RoleRequest $request, Role $role)  
 {  
 $role->update($request->validated());  
  
 return $role;  
 }  
  
 *//Удаляет роль.* public function destroy(Role $role)  
 {  
 $role->delete();  
  
 return response()->json();  
 }  
}

Контроллер SearchController (см. лист. 40) реализует функционал поиска товаров по ключевому слову.

Метод search принимает параметр search из запроса, выполняет поиск продуктов, у которых в поле title или comment содержится указанная строка, с помощью оператора LIKE. Затем возвращает результат поиска в формате JSON с исходным запросом и списком найденных продуктов.

Данный контроллер обеспечивает простой и эффективный поиск товаров по названию и описанию.

Листинг 40 – Контроллер SearchController.

class SearchController extends Controller  
{  
 public function search(Request $request)  
 {  
 $query = $request->input('search'); *// важно: 'search', а не 'query'* $products = Product::*where*('title', 'like', '%' . $query . '%')  
 ->orWhere('comment', 'like', '%' . $query . '%')  
 ->get();  
  
 return response()->json([  
 'query' => $query,  
 'products' => $products  
 ]);  
 }  
}

Контроллер TypeCategoryController (см. лист. 41) отвечает за управление категориями типов продуктов, предоставляя полный набор CRUD-операций.

Метод index возвращает коллекцию всех категорий типов в виде ресурса TypeCategoryResource.

Метод store создаёт новую категорию типа на основе валидированных данных из TypeCategoryRequest.

Методы show, update и destroy обеспечивают просмотр, обновление и удаление конкретной категории типа соответственно.

Данный контроллер обеспечивает удобное и структурированное управление типами категорий в системе.

Листинг 41 – Контроллер TypeCategoryController.

class TypeCategoryController extends Controller  
{  
 *//Возвращает коллекцию всех категорий типов.* public function index()  
 {  
 return TypeCategoryResource::*collection*(TypeCategory::*all*());  
 }  
  
 *//Создает новую категорию типа.* public function store(TypeCategoryRequest $request)  
 {  
 return TypeCategory::*create*($request->validated());  
 }  
  
 *//Возвращает информацию о конкретной категории типа.* public function show(TypeCategory $typeCategory)  
 {  
 return $typeCategory;  
 }  
  
 *//Обновляет информацию о конкретной категории типа.* public function update(TypeCategoryRequest $request, TypeCategory $typeCategory)  
 {  
 $typeCategory->update($request->validated());  
  
 return $typeCategory;  
 }  
  
 *//Удаляет категорию типа.* public function destroy(TypeCategory $typeCategory)  
 {  
 $typeCategory->delete();  
  
 return response()->json();  
 }  
}

## **Панель администратора Filament**

Для упрощения работы администратору был добавлен плагин Filament, который предоставляет доступ к админ-панели. Расположение и редактирование плагина происходит по пути app/Filament.

***Провайдер админ-панели***

Авторизация, обнаружение и пути прописаны в файле AdminPanelProvider, который находится по пути app/Providers/Filament/.

Класс AdminPanelProvider наследуется от PanelProvider и отвечает за создание и настройку панели администратора Filament.

В методе panel задаются основные параметры панели: уникальный идентификатор, путь доступа (admin), включение страницы входа, цветовая схема, иконка favicon и логотип, название бренда.

Также реализовано автоматическое обнаружение ресурсов, страниц и виджетов Filament в соответствующих папках приложения.

Определён набор страниц и виджетов, включаемых в панель.

Для панели и аутентификации прописан список необходимых промежуточных HTTP-слоёв (middleware), обеспечивающих безопасность и корректную работу панели.

Данный класс централизует все настройки и обеспечивает полноценное управление админ-панелью (см. листинг 42).

Листинг 42 – авторизация в админ-панели.

class AdminPanelProvider extends PanelProvider  
{  
 */\*\*  
 \* @throws \Exception  
 \*/  
  
 //Создает и настраивает панель администратора Filament.* public function panel(Panel $panel): Panel  
 {  
 return $panel  
 ->default()  
 ->id('admin') *// Уникальный идентификатор панели* ->path('admin') *// Путь к панели* ->login() *// Включает страницу входа* ->colors([ *// Настройка цветовой схемы панели* 'primary' => Color::*Amber*,  
 ])  
 ->favicon(asset('images/logo.jpg')) *// Путь к иконке favicon* ->brandLogo(asset('images/logo.jpg')) *// Путь к логотипу* ->brandName(asset('MotoShop')) *// Название бренда* ->discoverResources(in: app\_path('Filament/Resources'), for: 'App\\Filament\\Resources') *// Обнаружение ресурсов Filament* ->discoverPages(in: app\_path('Filament/Pages'), for: 'App\\Filament\\Pages') *// Обнаружение страниц Filament* ->pages([  
 *// Добавление страниц в панель* Pages\Dashboard::class,  
 ])  
 ->discoverWidgets(in: app\_path('Filament/Widgets'), for: 'App\\Filament\\Widgets') *// Обнаружение виджетов Filament* ->widgets([  
 *// Добавление виджетов в панель* Widgets\AccountWidget::class,  
 Widgets\FilamentInfoWidget::class,  
 ])  
 ->middleware([  
 *// Промежуточные слои HTTP для панели* EncryptCookies::class,  
 AddQueuedCookiesToResponse::class,  
 StartSession::class,  
 AuthenticateSession::class,  
 ShareErrorsFromSession::class,  
 VerifyCsrfToken::class,  
 SubstituteBindings::class,  
 DisableBladeIconComponents::class,  
 DispatchServingFilamentEvent::class,  
 ])  
 ->authMiddleware([  
 *// Промежуточные слои аутентификации для панели* Authenticate::class,  
 ]);  
 }  
}

***Виджеты***

Статистика блоки на главной странице написана в виджете StatsAdminOverview (см. рисунок 48).

Класс StatsAdminOverview наследуется от BaseWidget и отвечает за отображение основной статистики на главной странице административной панели. Метод getStats возвращает массив статистических данных, включая количество зарегистрированных пользователей, созданных заказов и доступных товаров на сайте. Каждый статистический элемент оформлен с описанием, иконкой и цветовой схемой для наглядного представления информации (см. листинг 43).

Листинг 43 – основная статистика на главной странице.

class StatsAdminOverview extends *BaseWidget*{  
 *//Получает статистику для административного обзорного виджета.* protected function getStats(): array  
 {  
 return [  
 *// Статистика по количеству зарегистрированных пользователей* Stat::*make*('Пользователи', User::*query*()->count())  
 ->description('Зарегистрированных пользователей')  
 ->descriptionIcon('heroicon-m-users')  
 ->chart([7, 2, 10, 3, 15, 4, 17])  
 ->color('success'),  
  
 *// Статистика по количеству заказов* Stat::*make*('Заказы', Cart::*query*()->count())  
 ->description('Всего созданных заказов')  
 ->descriptionIcon('heroicon-m-banknotes')  
 ->color('info'),  
  
 *// Статистика по количеству товаров на сайте* Stat::*make*('Товаров на сайте', Product::*query*()->count())  
 ->description('Доступных товаров')  
 ->descriptionIcon('heroicon-m-cube')  
 ->color('primary'),  
 ];  
  
 }  
}

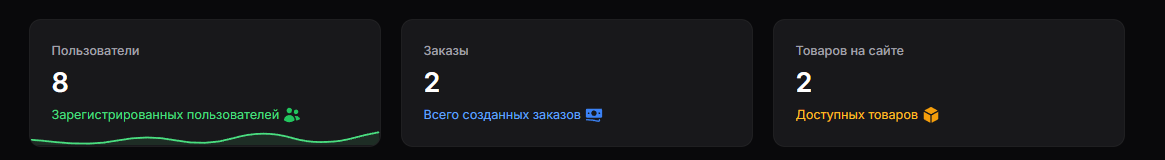


Рисунок 48 – Блоки статистики

Статистика графиком написана в виджете UserStatsAdminChart (см. рисунок 49).

Класс UsersStatsAdminChart наследуется от ChartWidget и предназначен для отображения статистики зарегистрированных пользователей сайта в виде графика. Виджет имеет заголовок, цветовую схему и занимает определённое количество столбцов в макете панели. В методе getData формируются данные для графика на основе количества регистраций пользователей за выбранный период, с возможностью фильтрации по дате — сегодня, неделя, месяц или год. Тип графика задан как линейный, что позволяет наглядно отслеживать динамику регистрации пользователей в течение времени (см. листинг 44).

Листинг 44 – статистика зарег. пользователей сайта.

class UsersStatsAdminChart extends ChartWidget  
{  
 *//Заголовок виджета.* protected static ?string *$heading* = 'Статистика';  
  
 *//Цвет виджета.* protected static string *$color* = 'info';  
  
 *//Количество столбцов, занимаемых виджетом в макете.* protected int | string | array $columnSpan = [  
 'md' => 2,  
 'xl' => 2,  
 ];  
  
 *//Фильтр данных для графика.* public ?string $filter = 'year';  
  
 *//Получает данные для построения графика.* protected function getData(): array  
 {  
 $data = Trend::*model*(User::class)  
 ->between(  
 start: now()->startOfYear(),  
 end: now()->endOfYear(),  
 )  
 ->perMonth()  
 ->count();  
 $activeFilter = $this->filter;  
 return [  
 'datasets' => [  
 [  
 'label' => 'Пользователей зарегистрировалось',  
 'data' => $data->map(fn (TrendValue $value) => $value->aggregate),  
 ],  
 ],  
 'labels' => $data->map(fn (TrendValue $value) => $value->date),  
 ];  
 }  
  
 *//Получает доступные фильтры для выбора диапазона данных на графике.* protected function getFilters(): ?array  
 {  
 return [  
 'today' => 'За сегодня',  
 'week' => 'За неделю',  
 'month' => 'За месяц',  
 'year' => 'За год',  
 ];  
 }  
  
 *//Возвращает тип графика.* protected function getType(): string  
 {  
 return 'line';  
 }  
}



Рисунок 49 – Статистика в графике

***Ресурс модели Filament***

Отображение и работоспособность кнопок описана в Resource моделях по пути app/Filament/Resources.

Далее будет предоставлен код с подробными комментариями, которые описывают каждый метод.

Класс CartResource наследуется от Resource и служит для управления ресурсом корзины в админ-панели. В нём указана связанная модель Cart, а также заданы иконка, метка и группа навигации для удобного отображения в интерфейсе. Определена форма для создания и редактирования записей с обязательными числовыми полями order\_id и product\_id. В таблице ресурса настроены колонки для отображения идентификатора заказа, названия продукта и временных меток создания и обновления, с возможностью сортировки и скрытия некоторых колонок. Реализованы действия просмотра, редактирования и массового удаления записей. Также заданы маршруты для страниц списка, создания, просмотра и редактирования записей. Ресурс обеспечивает полный набор инструментов для работы с корзиной в административной части приложения (см. листинг 45).

Листинг 45 – ресурс CartResource

class CartResource extends Resource  
{  
 *// Указание модели, связанной с ресурсом.* protected static ?string *$model* = Cart::class;  
  
 *// Иконка для навигации к ресурсу.* protected static ?string *$navigationIcon* = 'heroicon-o-shopping-cart';  
  
 *// Метка для навигации к ресурсу.* protected static ?string *$navigationLabel* = 'Корзина';  
  
 *// Группа для навигации к ресурсу.* protected static ?string *$navigationGroup* = 'Управление корзиной';  
  
 *// Определение формы для создания и редактирования записей.* public static function form(Form $form): Form  
 {  
 return $form  
 ->schema([  
 Forms\Components\TextInput::*make*('order\_id')  
 ->required()  
 ->numeric(),  
 Forms\Components\TextInput::*make*('product\_id')  
 ->required()  
 ->numeric(),  
 ]);  
 }  
  
 *// Определение таблицы для отображения записей.* public static function table(Table $table): Table  
 {  
 return $table  
 ->columns([  
 Tables\Columns\TextColumn::*make*('order\_id')  
 ->numeric()  
 ->sortable(),  
 Tables\Columns\TextColumn::*make*('product.title')  
 ->sortable(),  
 Tables\Columns\TextColumn::*make*('created\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 Tables\Columns\TextColumn::*make*('updated\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 ])  
 ->filters([  
 *//* ])  
 ->actions([  
 Tables\Actions\ViewAction::*make*(),  
 Tables\Actions\EditAction::*make*(),  
 ])  
 ->bulkActions([  
 Tables\Actions\BulkActionGroup::*make*([  
 Tables\Actions\DeleteBulkAction::*make*(),  
 ]),  
 ]);  
 }  
  
 *// Получение связанных ресурсов.* public static function getRelations(): array  
 {  
 return [  
 *// Здесь могут быть определены связанные ресурсы* ];  
 }  
  
 *// Получение страниц ресурса.* public static function getPages(): array  
 {  
 return [  
 'index' => Pages\ListCarts::*route*('/'),  
 'create' => Pages\CreateCart::*route*('/create'),  
 'view' => Pages\ViewCart::*route*('/{record}'),  
 'edit' => Pages\EditCart::*route*('/{record}/edit'),  
 ];  
 }  
}

Класс OrderResource наследуется от Resource и предназначен для управления ресурсом заказов в административной панели. В нём указана связанная модель Order, а также заданы иконка, метка и группа навигации для удобного отображения в интерфейсе. Определена форма для создания и редактирования записей с обязательными числовыми полями total\_price и user\_id. В таблице ресурса настроены колонки для отображения общей стоимости заказа, идентификатора пользователя и временных меток создания и обновления, с возможностью сортировки и скрытия некоторых колонок. Реализованы действия просмотра, редактирования и массового удаления записей. Заданы маршруты для страниц списка, создания, просмотра и редактирования заказов. Ресурс обеспечивает полный набор инструментов для работы с заказами в административной части приложения (см. листинг 46).

Листинг 46 – ресурс OrderResource

class OrderResource extends Resource  
{  
 *// Модель, связанная с ресурсом.* protected static ?string *$model* = Order::class;  
  
 *// Иконка для навигации к ресурсу.* protected static ?string *$navigationIcon* = 'heroicon-o-archive-box-arrow-down';  
  
 *// Метка для навигации к ресурсу.* protected static ?string *$navigationLabel* = 'Заказ';  
  
 *// Группа для навигации к ресурсу.* protected static ?string *$navigationGroup* = 'Управление корзиной';  
  
 *// Определяет форму для создания и редактирования записей.* public static function form(Form $form): Form  
 {  
 return $form  
 ->schema([  
 Forms\Components\TextInput::*make*('total\_price')  
 ->required()  
 ->numeric(),  
 Forms\Components\TextInput::*make*('user\_id')  
 ->required()  
 ->numeric(),  
 ]);  
 }  
  
 *// Определяет таблицу для отображения записей.* public static function table(Table $table): Table  
 {  
 return $table  
 ->columns([  
 Tables\Columns\TextColumn::*make*('total\_price')  
 ->numeric()  
 ->sortable(),  
 Tables\Columns\TextColumn::*make*('user\_id')  
 ->numeric()  
 ->sortable(),  
 Tables\Columns\TextColumn::*make*('created\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 Tables\Columns\TextColumn::*make*('updated\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 ])  
 ->filters([  
 *//* ])  
 ->actions([  
 Tables\Actions\ViewAction::*make*(),  
 Tables\Actions\EditAction::*make*(),  
 ])  
 ->bulkActions([  
 Tables\Actions\BulkActionGroup::*make*([  
 Tables\Actions\DeleteBulkAction::*make*(),  
 ]),  
 ]);  
 }  
  
 *// Получает связанные ресурсы.* public static function getRelations(): array  
 {  
 return [  
 *// Здесь могут быть определены связанные ресурсы* ];  
 }  
  
 *// Получает страницы ресурса.* public static function getPages(): array  
 {  
 return [  
 'index' => Pages\ListOrders::*route*('/'),  
 'create' => Pages\CreateOrder::*route*('/create'),  
 'view' => Pages\ViewOrder::*route*('/{record}'),  
 'edit' => Pages\EditOrder::*route*('/{record}/edit'),  
 ];  
 }  
}

Класс ProductResource наследуется от Resource и предназначен для управления ресурсом товаров в административной панели. В нём указана связанная модель Product, а также заданы иконка, метка и группа навигации для удобного отображения в интерфейсе. Определена форма для создания и редактирования записей с обязательными полями: названием, бесплатной загрузкой, комментарием, ценой с префиксом рубля, годом и выбором категории с использованием связи с типом категории. Таблица ресурса содержит колонки для отображения названия, комментария, цены с форматированием, года, названия категории и временных меток создания и обновления с возможностью сортировки, поиска и скрытия некоторых колонок. Реализованы действия просмотра, редактирования и массового удаления записей. Заданы маршруты для страниц списка, создания, просмотра и редактирования товаров. Ресурс предоставляет полный функционал для управления товарами в админ-панели (см. листинг 47).

Листинг 47 – ресурс ProductResource.

class ProductResource extends Resource  
{  
 *// Модель, связанная с ресурсом.* protected static ?string *$model* = Product::class;  
  
 *// Иконка для навигации к ресурсу.* protected static ?string *$navigationIcon* = 'heroicon-o-truck';  
  
 *// Метка для навигации к ресурсу.* protected static ?string *$navigationLabel* = 'Товары';  
  
 *// Группа для навигации к ресурсу.* protected static ?string *$navigationGroup* = 'Управление товарами';  
  
 *// Определяет форму для создания и редактирования записей.* public static function form(Form $form): Form  
 {  
 return $form  
 ->schema([  
 Forms\Components\TextInput::*make*('title')  
 ->required()  
 ->maxLength(255),  
 Forms\Components\TextInput::*make*('download\_free')  
 ->required()  
 ->nullable(),  
 Forms\Components\TextInput::*make*('comment')  
 ->required()  
 ->maxLength(255),  
 Forms\Components\TextInput::*make*('price')  
 ->required()  
 ->numeric()  
 ->prefix('₽'),  
 Forms\Components\TextInput::*make*('year')  
 ->required(),  
 Forms\Components\Select::*make*('type\_category\_id')  
 ->required()  
 ->relationship('category', 'name')  
 ->native(false)  
 ]);  
 }  
  
 */\*\*  
 \* @throws \Exception  
 \*/  
 // Определяет таблицу для отображения записей.* public static function table(Table $table): Table  
 {  
 return $table  
 ->columns([  
 Tables\Columns\TextColumn::*make*('title')  
 ->searchable(),  
 Tables\Columns\TextColumn::*make*('comment')  
 ->searchable(),  
 Tables\Columns\TextColumn::*make*('price')  
 ->money()  
 ->sortable(),  
 Tables\Columns\TextColumn::*make*('year'),  
 Tables\Columns\TextColumn::*make*('category.name')  
 ->sortable(),  
 Tables\Columns\TextColumn::*make*('created\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 Tables\Columns\TextColumn::*make*('updated\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 ])  
 ->filters([  
 *//SelectFilter::make('Product')  
 //->relationship('product', 'title')* ])  
 ->actions([  
 Tables\Actions\ViewAction::*make*(),  
 Tables\Actions\EditAction::*make*(),  
 ])  
 ->bulkActions([  
 Tables\Actions\BulkActionGroup::*make*([  
 Tables\Actions\DeleteBulkAction::*make*(),  
 ]),  
 ]);  
 }  
  
 *// Получает связанные ресурсы.* public static function getRelations(): array  
 {  
 return [  
 *//* ];  
 }  
  
 *// Получает страницы ресурса.* public static function getPages(): array  
 {  
 return [  
 'index' => Pages\ListProducts::*route*('/'),  
 'create' => Pages\CreateProduct::*route*('/create'),  
 'view' => Pages\ViewProduct::*route*('/{record}'),  
 'edit' => Pages\EditProduct::*route*('/{record}/edit'),  
 ];  
 }  
}

Класс RoleResource наследуется от Resource и предназначен для управления ролями пользователей в административной панели. В нём указана связанная модель Role, а также заданы иконка, метка и группа навигации для удобного отображения в меню. Определена форма для создания и редактирования записей с обязательными полями имени и кода роли, ограниченными по длине. Таблица ресурса содержит колонки для отображения имени, кода и временных меток создания и обновления, с возможностью поиска, сортировки и скрытия некоторых колонок по умолчанию. Реализованы действия просмотра, редактирования и массового удаления записей. Заданы маршруты для страниц списка, создания, просмотра и редактирования ролей. Ресурс обеспечивает полный функционал управления ролями в админ-панели (см. листинг 48).

Листинг 48 – ресурс RoleResource.

class RoleResource extends Resource  
{  
 *// Модель, связанная с ресурсом.* protected static ?string *$model* = Role::class;  
  
 *// Иконка для навигации к ресурсу.* protected static ?string *$navigationIcon* = 'heroicon-o-adjustments-vertical';  
  
 *// Метка для навигации к ресурсу.* protected static ?string *$navigationLabel* = 'Роли';  
  
 *// Группа для навигации к ресурсу.* protected static ?string *$navigationGroup* = 'Управление пользователями';  
  
 *// Определяет форму для создания и редактирования записей.* public static function form(Form $form): Form  
 {  
 return $form  
 ->schema([  
 Forms\Components\TextInput::*make*('name')  
 ->required()  
 ->maxLength(255),  
 Forms\Components\TextInput::*make*('code')  
 ->required()  
 ->maxLength(255),  
 ]);  
 }  
  
 *// Определяет таблицу для отображения записей.* public static function table(Table $table): Table  
 {  
 return $table  
 ->columns([  
 Tables\Columns\TextColumn::*make*('name')  
 ->searchable(),  
 Tables\Columns\TextColumn::*make*('code')  
 ->searchable(),  
 Tables\Columns\TextColumn::*make*('created\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 Tables\Columns\TextColumn::*make*('updated\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 ])  
 ->filters([  
 *//* ])  
 ->actions([  
 Tables\Actions\ViewAction::*make*(),  
 Tables\Actions\EditAction::*make*(),  
 ])  
 ->bulkActions([  
 Tables\Actions\BulkActionGroup::*make*([  
 Tables\Actions\DeleteBulkAction::*make*(),  
 ]),  
 ]);  
 }  
  
 *// Получает связанные ресурсы.* public static function getRelations(): array  
 {  
 return [  
 *// Здесь могут быть определены связанные ресурсы* ];  
 }  
  
 *// Получает страницы ресурса.* public static function getPages(): array  
 {  
 return [  
 'index' => Pages\ListRoles::*route*('/'),  
 'create' => Pages\CreateRole::*route*('/create'),  
 'view' => Pages\ViewRole::*route*('/{record}'),  
 'edit' => Pages\EditRole::*route*('/{record}/edit'),  
 ];  
 }  
}

Класс TypeCategoryResource наследуется от Resource и отвечает за управление категориями товаров в административной панели. В нем задана связанная модель TypeCategory, а также иконка, метка и группа навигации для отображения в меню. Определена форма для создания и редактирования категорий с обязательным полем имени и ограничением длины. Таблица ресурса включает колонки для имени и временных меток создания и обновления, которые можно сортировать, искать и скрывать по умолчанию. Реализованы действия просмотра, редактирования и массового удаления записей. Определены маршруты для страниц списка, создания, просмотра и редактирования категорий. Этот ресурс обеспечивает полный функционал для управления категориями товаров в админ-панели (см. листинг 49).

Листинг 49 – ресурс TypeCategoryResource.

class TypeCategoryResource extends Resource  
{  
 *// Модель, связанная с ресурсом.* protected static ?string *$model* = TypeCategory::class;  
  
 *// Иконка для навигации к ресурсу.* protected static ?string *$navigationIcon* = 'heroicon-o-tag';  
  
 *// Метка для навигации к ресурсу.* protected static ?string *$navigationLabel* = 'Категории';  
  
 *// Группа для навигации к ресурсу.* protected static ?string *$navigationGroup* = 'Управление товарами';  
  
 *// Определяет форму для создания и редактирования записей.* public static function form(Form $form): Form  
 {  
 return $form  
 ->schema([  
 Forms\Components\TextInput::*make*('name')  
 ->required()  
 ->maxLength(255),  
 ]);  
 }  
  
 *// Определяет таблицу для отображения записей.* public static function table(Table $table): Table  
 {  
 return $table  
 ->columns([  
 Tables\Columns\TextColumn::*make*('name')  
 ->searchable(),  
 Tables\Columns\TextColumn::*make*('created\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 Tables\Columns\TextColumn::*make*('updated\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 ])  
 ->filters([  
 *//* ])  
 ->actions([  
 Tables\Actions\ViewAction::*make*(),  
 Tables\Actions\EditAction::*make*(),  
 ])  
 ->bulkActions([  
 Tables\Actions\BulkActionGroup::*make*([  
 Tables\Actions\DeleteBulkAction::*make*(),  
 ]),  
 ]);  
 }  
  
 *// Получает связанные ресурсы.* public static function getRelations(): array  
 {  
 return [  
 *// Здесь могут быть определены связанные ресурсы* ];  
 }  
  
 *// Получает страницы ресурса.* public static function getPages(): array  
 {  
 return [  
 'index' => Pages\ListTypeCategories::*route*('/'),  
 'create' => Pages\CreateTypeCategory::*route*('/create'),  
 'view' => Pages\ViewTypeCategory::*route*('/{record}'),  
 'edit' => Pages\EditTypeCategory::*route*('/{record}/edit'),  
 ];  
 }  
}

Класс UserResource наследуется от Resource и предназначен для управления пользователями в административной панели. В нем указана связанная модель User, иконка, метка и группа навигации для удобного отображения в меню. В форме создания и редактирования предусмотрены поля для имени, email, логина, пароля с хэшированием перед сохранением, загрузки изображения пользователя и выбора роли через связь с моделью Role. В таблице отображаются колонки с основными данными пользователя: имя, email, логин, изображение, роль и временные метки создания и обновления, с возможностью поиска, сортировки и скрытия некоторых колонок. Реализованы действия просмотра, редактирования и массового удаления пользователей. Определены маршруты для страниц списка, создания, просмотра и редактирования записей. Этот ресурс обеспечивает полный набор функций для работы с пользователями в админ-панели (см. листинг 50).

Листинг 50 – ресурс UserResource.

class UserResource extends Resource  
{  
 *//Модель, связанная с ресурсом.* protected static ?string *$model* = User::class;  
  
 *//Иконка для навигации к ресурсу.* protected static ?string *$navigationIcon* = 'heroicon-o-user-circle';  
  
 *//Метка для навигации к ресурсу.* protected static ?string *$navigationLabel* = 'Пользователи';  
  
 *//Группа для навигации к ресурсу.* protected static ?string *$navigationGroup* = 'Управление пользователями';  
  
 *//Определяет форму для создания и редактирования записей.* public static function form(Form $form): Form  
 {  
  
 return $form  
 ->schema([  
 Forms\Components\TextInput::*make*('name')  
 ->required()  
 ->maxLength(255),  
 Forms\Components\TextInput::*make*('email')  
 ->email()  
 ->required()  
 ->maxLength(255),  
 Forms\Components\TextInput::*make*('login')  
 ->required()  
 ->maxLength(255),  
 Forms\Components\TextInput::*make*('password')  
 ->password()  
 ->required()  
 ->maxLength(255),  
 Forms\Components\FileUpload::*make*('image')  
 ->image()  
 ->disk('public'),  
 Forms\Components\Select::*make*('role\_id')  
 ->required()  
 ->relationship('role', 'name')  
 ->native(false)  
 ]);  
 }  
  
 *//Определяет таблицу для отображения записей.* public static function table(Table $table): Table  
 {  
 return $table  
 ->columns([  
 Tables\Columns\TextColumn::*make*('name')  
 ->searchable(),  
 Tables\Columns\TextColumn::*make*('email')  
 ->searchable(),  
 Tables\Columns\TextColumn::*make*('login')  
 ->searchable(),  
 Tables\Columns\ImageColumn::*make*('image'),  
 Tables\Columns\TextColumn::*make*('role\_id')  
 ->numeric()  
 ->sortable(),  
 Tables\Columns\TextColumn::*make*('created\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 Tables\Columns\TextColumn::*make*('updated\_at')  
 ->dateTime()  
 ->sortable()  
 ->toggleable(isToggledHiddenByDefault: true),  
 ])  
 ->filters([  
 *//* ])  
 ->actions([  
 Tables\Actions\ViewAction::*make*(),  
 Tables\Actions\EditAction::*make*(),  
 ])  
 ->bulkActions([  
 Tables\Actions\BulkActionGroup::*make*([  
 Tables\Actions\DeleteBulkAction::*make*(),  
 ]),  
 ]);  
 }  
  
 *//Получает связанные ресурсы.* public static function getRelations(): array  
 {  
 return [  
 *// Здесь могут быть определены связанные ресурсы* ];  
 }  
  
 *//Получает страницы ресурса.* public static function getPages(): array  
 {  
 return [  
 'index' => Pages\ListUsers::*route*('/'),  
 'create' => Pages\CreateUser::*route*('/create'),  
 'view' => Pages\ViewUser::*route*('/{record}'),  
 'edit' => Pages\EditUser::*route*('/{record}/edit'),  
 ];  
 }  
  
 *//Изменяет данные перед созданием записи.* protected function mutateFormDataBeforeCreate(array $data): array  
 {  
 $data['password'] = Hash::make($data['password']); *//* return $data;  
 }  
}

***Контроллеры Filament***

Ниже будут предоставлены контроллеры, которые выполняют функции просмотра, создания и редактирования.

Контроллеры CreateCart, EditCart, ListCart, ViewCart

Класс CreateCart наследуется от CreateRecord и служит контроллером для страницы создания записи в ресурсе CartResource. Он связывает страницу создания с конкретным ресурсом корзины, обеспечивая стандартный функционал добавления новой записи в админ-панели (см. листинг 51).

Листинг 51 – CreateCart controller.

class CreateCart extends CreateRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = CartResource::class;  
}

Класс EditCart наследуется от EditRecord и представляет контроллер страницы редактирования записи ресурса CartResource. Он связывает страницу с ресурсом корзины и расширяет стандартные действия заголовка страницы, добавляя кнопки просмотра и удаления записи. Метод save переопределён для обработки сохранения данных формы: сначала получает состояние формы, обновляет модель записи, сохраняет её, а затем по необходимости перенаправляет пользователя на страницу списка записей. Таким образом, класс обеспечивает удобное управление редактированием корзин в админ-панели (см. листинг 52).

Листинг 52 – EditCart controller.

class EditCart extends EditRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = CartResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\ViewAction::*make*(), *// Действие "Просмотр"* Actions\DeleteAction::*make*(), *// Действие "Удаление"* ];  
 }  
  
 *// Метод сохранения записи.* public function save(bool $shouldRedirect = true, bool $shouldSendSavedNotification = true): void  
 {  
 $data = $this->form->getState();  
  
 *// Примените изменения к данным* $this->record->update($data);  
  
 *// Сохраните модель, если нужно* $this->record->save();  
  
 *// Опционально: выполните дополнительные действия после сохранения* if ($shouldRedirect) {  
 *// Перенаправление пользователя после сохранения* $this->redirect($this->getResource()::*getUrl*('index'));  
 }  
 }  
}

Класс ListCarts наследуется от ListRecords и представляет контроллер страницы списка записей ресурса CartResource. Он устанавливает связь страницы с ресурсом корзины и расширяет стандартные действия заголовка страницы, добавляя кнопку создания новой записи. Таким образом, класс обеспечивает возможность отображения списка корзин с функционалом добавления новых элементов в админ-панели (см. листинг 53).

Листинг 53 – ListCart controller.

class ListCarts extends ListRecords  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = CartResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\CreateAction::*make*(), *// Действие "Создать"* ];  
 }  
}

Класс ViewCart наследуется от ViewRecord и представляет контроллер страницы просмотра записи ресурса CartResource. Он связывает страницу с ресурсом корзины и расширяет стандартные действия заголовка страницы, добавляя кнопку редактирования записи. Таким образом, класс обеспечивает удобный просмотр деталей корзины с возможностью перехода к её редактированию в админ-панели (см. листинг 54).

Листинг 54 – ViewCart controller.

class ViewCart extends ViewRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = CartResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\EditAction::*make*(), *// Действие "Редактировать"* ];  
 }  
}

Контроллеры CreateOrder, EditOrder, ListOrder, ViewOrder

Класс CreateOrder наследуется от CreateRecord и представляет контроллер страницы создания записи ресурса OrderResource. Он устанавливает связь страницы с ресурсом заказов, обеспечивая базовый функционал для создания новых заказов в админ-панели без добавления дополнительных действий или переопределений (см. листинг 55).

Листинг 55 – CreateOrder controller.

class CreateOrder extends CreateRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = OrderResource::class;  
}

Класс EditOrder наследуется от EditRecord и представляет контроллер страницы редактирования записи ресурса OrderResource. Он связывает страницу с ресурсом заказов и расширяет стандартные действия заголовка страницы, добавляя кнопки просмотра и удаления записи. Метод save переопределён для обработки сохранения данных формы: сначала получает состояние формы, обновляет модель записи, сохраняет её, а затем при необходимости перенаправляет пользователя на страницу списка заказов. Таким образом, класс обеспечивает удобное управление редактированием заказов в админ-панели (см. листинг 56).

Листинг 56 – EditOrder controller.

class EditOrder extends EditRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = OrderResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\ViewAction::*make*(), *// Действие "Просмотр"* Actions\DeleteAction::*make*(), *// Действие "Удаление"* ];  
 }  
  
 *// Метод сохранения записи.* public function save(bool $shouldRedirect = true, bool $shouldSendSavedNotification = true): void  
 {  
 $data = $this->form->getState();  
  
 *// Примените изменения к данным* $this->record->update($data);  
  
 *// Сохраните модель, если нужно* $this->record->save();  
  
 *// Опционально: выполните дополнительные действия после сохранения* if ($shouldRedirect) {  
 *// Перенаправление пользователя после сохранения* $this->redirect($this->getResource()::*getUrl*('index'));  
 }  
 }  
}

Класс ListOrders наследуется от ListRecords и представляет контроллер страницы списка записей ресурса OrderResource. Он устанавливает связь страницы с ресурсом заказов и расширяет стандартные действия заголовка страницы, добавляя кнопку создания новой записи. Таким образом, класс обеспечивает возможность отображения списка заказов с функционалом добавления новых элементов в админ-панели (см. листинг 57).

Листинг 57 – ListOrder controller.

class ListOrders extends ListRecords  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = OrderResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\CreateAction::*make*(), *// Действие "Создать"* ];  
 }  
}

Класс ViewOrder наследуется от ViewRecord и представляет контроллер страницы просмотра записи ресурса OrderResource. Он связывает страницу с ресурсом заказов и расширяет стандартные действия заголовка страницы, добавляя кнопку редактирования записи. Таким образом, класс обеспечивает удобный просмотр деталей заказа с возможностью перехода к его редактированию в админ-панели (см. листинг 58).

Листинг 58 – ViewOrder controller.

class ViewOrder extends ViewRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = OrderResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\EditAction::*make*(), *// Действие "Редактировать"* ];  
 }  
}

Контроллеры CreateProduct, EditProduct, ListProduct, ViewProduct

Класс CreateProduct наследуется от CreateRecord и представляет контроллер страницы создания записи ресурса ProductResource. Он устанавливает связь страницы с ресурсом продуктов и определяет схему формы, включающую поля для названия продукта, ссылки на скачивание, комментария, цены с валидацией, года выпуска и выбора категории с использованием связи. Для загрузки изображений предусмотрена отдельная группа с повторителем, позволяющим добавлять несколько файлов. Метод handleRecordCreation отвечает за создание новой записи продукта и сохранение связанных с ней изображений. Таким образом, класс обеспечивает полноценный интерфейс для добавления продуктов с их детальной информацией и мультимедийными файлами в админ-панели (см. листинг 59).

Листинг 59 – CreateProduct controller.

class CreateProduct extends CreateRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = ProductResource::class;  
  
 *// Схема формы* public function form(Form $form): Form  
 {  
 return $form  
 ->schema([  
 TextInput::*make*('title')->label('Название продукта')->required(),  
 TextInput::*make*('download\_free')->label('Ссылка на скачивание')->required(),  
 Textarea::*make*('comment')->label('Комментарий')->required(),  
 TextInput::*make*('price')->label('Цена')->numeric()  
 ->required()->rules('numeric|max:9999.00|min:0|required')->prefix('₽'),  
 TextInput::*make*('year')->label('Год')->required(),  
 Select::*make*('type\_category\_id')->label('Категория')->required()  
 ->relationship('category', 'name')->native(false),  
  
  
 Fieldset::*make*('Изображения')->columnSpan(2)  
 ->schema([  
 Repeater::*make*('images')->label('Изображение')  
 ->schema([  
 FileUpload::*make*('image')->image()->required(),  
 ])  
 ->createItemButtonLabel('Добавить изображение')  
 ->columnSpan(2)->grid(2)->required(),  
 ]),  
 ]);  
 }  
  
 public function handleRecordCreation(array $data): \Illuminate\Database\Eloquent\Model  
 {  
 $product = static::*getModel*()::*create*($data);  
  
 $images = collect($data['images'])->map(function ($imageData) use ($product) {  
 return new ProductImage($imageData + ['product\_id' => $product->id]);  
 });  
 $product->images()->saveMany($images);  
  
 return $product;  
 }  
}

Класс EditProduct наследуется от EditRecord и представляет контроллер страницы редактирования записи ресурса ProductResource. Он устанавливает связь страницы с ресурсом продуктов и определяет схему формы, включающую поля для названия продукта, ссылки на скачивание, комментария, цены с валидацией, года выпуска и выбора категории, а также блок с повторителем для управления изображениями, позволяющим добавлять и удалять файлы с подтверждением удаления. В методе mount происходит инициализация состояния формы с загрузкой связанных изображений, а метод refreshState обновляет текущее состояние формы на основе данных из базы. Метод save переопределён для выполнения сохранения данных внутри транзакции: обновляются основные поля записи, добавляются новые изображения, при этом уведомления информируют пользователя об успешном сохранении. Метод deleteVolume отвечает за удаление выбранного изображения с последующим обновлением состояния формы и уведомлением. Заголовок страницы расширяется кнопками просмотра и удаления записи. Таким образом, класс обеспечивает комплексное управление редактированием продуктов с возможностью работы с мультимедийными файлами в админ-панели (см. листинг 60).

Листинг 60 – EditProduct controller.

class EditProduct extends EditRecord  
{  
 protected static string *$resource* = ProductResource::class;  
  
 public function mount($record): void  
 {  
 parent::*mount*($record);  
 $this->refreshState();  
 }  
  
 public function refreshState(): void  
 {  
 $images = Product::find($this->record->id)->images;  
  
 $state = $this->form->getRawState();  
 $state['images'] = $images ? $images->map->toArray()->toArray() : [];  
  
 $this->form->fill($state);  
 }  
  
 public function form(Form $form): Form  
 {  
 return $form  
 ->schema([  
 TextInput::*make*('title')->label('Название продукта')->required(),  
 TextInput::*make*('download\_free')->label('Ссылка на скачивание')->required(),  
 Textarea::*make*('comment')->label('Комментарий')->required(),  
 TextInput::*make*('price')->label('Цена')->numeric()  
 ->required()->rules('numeric|max:9999.00|min:0|required')->prefix('₽'),  
 TextInput::*make*('year')->label('Год')->required(),  
 Select::*make*('type\_category\_id')->label('Категория')->required()  
 ->relationship('category', 'name')->native(false),  
  
 Fieldset::*make*('Изображения')->columnSpan(2)  
 ->schema([  
 Repeater::*make*('images')->label('Изображение')  
 ->schema([  
 FileUpload::*make*('image')->image()->disk('public')->required(),  
 ])  
 ->createItemButtonLabel('Добавить изображение')  
 ->columnSpan(2)->grid(2)->deleteAction(  
 function (Action $action) {  
 $action->modalHeading("Вы действительно хотите удалить запись?")->action(function ($state, $arguments) {  
 if (isset($state[$arguments["item"]])) {  
 $this->deleteVolume($state[$arguments["item"]]);  
 }  
 });  
 }  
 )->required(),  
 ]),  
 ]);  
 }  
  
 protected function getHeaderActions(): array  
 {  
 return [  
 Actions\ViewAction::*make*(),  
 Actions\DeleteAction::*make*(),  
 ];  
 }  
  
 public function save(bool $shouldRedirect = true, bool $shouldSendSavedNotification = true): void  
 {  
 $data = $this->form->getState();  
  
 DB::transaction(function () use ($data) {  
 $this->record->update($data);  
  
 $existingImagePaths = $this->record->images->pluck('image')->toArray();  
  
 $imagesData = collect($data['images'])->unique('image')->values();  
 $newImages = $imagesData->reject(function ($imageData) use ($existingImagePaths) {  
 return in\_array($imageData['image'], $existingImagePaths);  
 });  
  
 $this->record->images()->saveMany($newImages->map(function ($imageData) {  
 return new ProductImage($imageData);  
 }));  
 });  
  
 Notification::*make*()  
 ->title('Изменения сохранены')  
 ->success()  
 ->send();  
  
 $this->refreshState();  
 }  
  
 public function deleteVolume($item): void  
 {  
 if (!isset($item['id'])) {  
 $this->refreshState();  
 return;  
 }  
  
 ProductImage::*find*($item['id'])->delete();  
 Notification::*make*()->title("Изображение удалено")->success()->send();  
 $this->refreshState();  
 }  
}

Класс ListProducts наследуется от ListRecords и представляет контроллер страницы списка записей ресурса ProductResource. Он устанавливает связь страницы с ресурсом продуктов и расширяет стандартные действия заголовка страницы, добавляя кнопку создания новой записи. Таким образом, класс обеспечивает возможность отображения списка продуктов с функционалом добавления новых элементов в админ-панели (см. листинг 61).

Листинг 61 – ListProduct controller.

class ListProducts extends ListRecords  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = ProductResource::class;  
  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\CreateAction::*make*(), *// Действие "Создать"* ];  
 }  
}

Класс ViewProduct наследуется от ViewRecord и представляет контроллер страницы просмотра записи ресурса ProductResource. Он связывает страницу с ресурсом продуктов и при инициализации загружает связанные изображения для отображения в форме. Схема формы включает поля для названия продукта, ссылки на скачивание, комментария, цены с валидацией, года выпуска, выбора категории и блок с повторителем для отображения изображений. Заголовок страницы расширяется кнопкой редактирования записи. Таким образом, класс обеспечивает удобный просмотр подробной информации о продукте вместе с его мультимедийными файлами в админ-панели (см. листинг 62).

Листинг 62 – ViewProduct controller.

class ViewProduct extends ViewRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = ProductResource::class;  
  
 public function mount($record): void  
 {  
 parent::*mount*($record);  
 $this->refreshState();  
 }  
  
 public function refreshState(): void  
 {  
 $images = Product::find($this->record->id)->images;  
  
 $state = $this->form->getRawState();  
 $state['images'] = $images ? $images->map->toArray()->toArray() : [];  
  
 $this->form->fill($state);  
 }  
  
 public function form(Form $form): Form  
 {  
 return $form  
 ->schema([  
 TextInput::*make*('title')->label('Название продукта')->required(),  
 TextInput::*make*('download\_free')->label('Ссылка на скачивание')->required(),  
 Textarea::*make*('comment')->label('Комментарий')->required(),  
 TextInput::*make*('price')->label('Цена')->numeric()  
 ->required()->rules('numeric|max:9999.00|min:0|required')->prefix('₽'),  
 TextInput::*make*('year')->label('Год')->required(),  
 Select::*make*('type\_category\_id')->label('Категория')->required()  
 ->relationship('category', 'name')->native(false),  
  
  
 Fieldset::*make*('Изображения')->columnSpan(2)  
 ->schema([  
 Repeater::*make*('images')->label('Изображение')  
 ->schema([  
 FileUpload::*make*('image')->image()->required(),  
 ])  
 ->createItemButtonLabel('Добавить изображение')  
 ->columnSpan(2)->grid(2)->required(),  
 ]),  
 ]);  
 }  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\EditAction::*make*(), *// Действие "Редактировать"* ];  
 }  
}

Контроллеры CreateRole, EditRole, ListRole, ViewRole

Класс CreateRole наследуется от CreateRecord и представляет контроллер страницы создания записи ресурса RoleResource. Он устанавливает связь страницы с ресурсом ролей, обеспечивая базовый функционал для создания новых ролей в админ-панели без дополнительных настроек или методов (см. листинг 63).

Листинг 63 – CreateRole controller.

class CreateRole extends CreateRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = RoleResource::class;  
}

Класс EditRole наследуется от EditRecord и представляет контроллер страницы редактирования записи ресурса RoleResource. Он связывает страницу с ресурсом ролей и расширяет стандартные действия заголовка страницы, добавляя кнопки просмотра и удаления записи. Метод save переопределён для обработки сохранения данных формы: сначала получает состояние формы, обновляет модель записи, сохраняет её, а затем при необходимости перенаправляет пользователя на страницу списка ролей. Таким образом, класс обеспечивает удобное управление редактированием ролей в админ-панели (см. листинг 64).

Листинг 64 – EditRole controller.

class EditRole extends EditRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = RoleResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\ViewAction::*make*(), *// Действие "Просмотр"* Actions\DeleteAction::*make*(), *// Действие "Удаление"* ];  
 }  
  
 *// Метод сохранения записи.* public function save(bool $shouldRedirect = true, bool $shouldSendSavedNotification = true): void  
 {  
 $data = $this->form->getState();  
  
 *// Примените изменения к данным* $this->record->update($data);  
  
 *// Сохраните модель, если нужно* $this->record->save();  
  
 *// Опционально: выполните дополнительные действия после сохранения* if ($shouldRedirect) {  
 $this->redirect($this->getResource()::*getUrl*('index'));  
 }  
 }  
}

Класс ListRoles наследуется от ListRecords и представляет контроллер страницы списка записей ресурса RoleResource. Он устанавливает связь страницы с ресурсом ролей и расширяет стандартные действия заголовка страницы, добавляя кнопку создания новой записи. Таким образом, класс обеспечивает возможность отображения списка ролей с функционалом добавления новых элементов в админ-панели (см. листинг 65).

Листинг 65 – ListRole controller.

class ListRoles extends ListRecords  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = RoleResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\CreateAction::*make*(), *// Действие "Создать"* ];  
 }  
}

Класс ViewRole наследуется от ViewRecord и представляет контроллер страницы просмотра записи ресурса RoleResource. Он связывает страницу с ресурсом ролей и расширяет стандартные действия заголовка страницы, добавляя кнопку редактирования записи. Таким образом, класс обеспечивает удобный просмотр деталей роли с возможностью перехода к её редактированию в админ-панели (см. листинг 66).

Листинг 66 – ViewRole controller.

class ViewRole extends ViewRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = RoleResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\EditAction::*make*(), *// Действие "Редактировать"* ];  
 }  
}

Контроллеры CreateTypeCategory, EditTypeCategory, ListTypeCategory, ViewTypeCategory

Класс CreateTypeCategory наследуется от CreateRecord и представляет контроллер страницы создания записи ресурса TypeCategoryResource. Он устанавливает связь страницы с ресурсом категорий типов, обеспечивая базовый функционал для создания новых категорий в админ-панели без дополнительных настроек или методов (см. листинг 67).

Листинг 67 – CreateTypeCategory controller.

class CreateTypeCategory extends CreateRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = TypeCategoryResource::class;  
}

Класс EditTypeCategory наследуется от EditRecord и представляет контроллер страницы редактирования записи ресурса TypeCategoryResource. Он связывает страницу с ресурсом категорий типов и расширяет стандартные действия заголовка страницы, добавляя кнопки просмотра и удаления записи. Метод save переопределён для обработки сохранения данных формы: сначала получает состояние формы, обновляет модель записи, сохраняет её, а затем при необходимости перенаправляет пользователя на страницу списка категорий типов. Таким образом, класс обеспечивает удобное управление редактированием категорий типов в админ-панели (см. листинг 68).

Листинг 68 – EditTypeCategory controller.

class EditTypeCategory extends EditRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = TypeCategoryResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\ViewAction::*make*(), *// Действие "Просмотр"* Actions\DeleteAction::*make*(), *// Действие "Удаление"* ];  
 }  
 public function save(bool $shouldRedirect = true, bool $shouldSendSavedNotification = true): void  
 {  
 $data = $this->form->getState();  
  
 *// Примените изменения к данным* $this->record->update($data);  
  
 *// Сохраните модель, если нужно* $this->record->save();  
  
 *// Опционально: выполните дополнительные действия после сохранения* if ($shouldRedirect) {  
 *// Перенаправление пользователя после сохранения* $this->redirect($this->getResource()::*getUrl*('index'));  
 }  
 }  
}

Класс ListTypeCategories наследуется от ListRecords и представляет контроллер страницы списка записей ресурса TypeCategoryResource. Он устанавливает связь страницы с ресурсом категорий типов и расширяет стандартные действия заголовка страницы, добавляя кнопку создания новой записи. Таким образом, класс обеспечивает возможность отображения списка категорий типов с функционалом добавления новых элементов в админ-панели (см. листинг 69).

Листинг 69 – ListTypeCategory controller.

class ListTypeCategories extends ListRecords  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = TypeCategoryResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\CreateAction::*make*(), *// Действие "Создать"* ];  
 }  
}

Класс ViewTypeCategory наследуется от ViewRecord и представляет контроллер страницы просмотра записи ресурса TypeCategoryResource. Связь с ресурсом устанавливается через статическое свойство $resource.

Метод getHeaderActions добавляет в заголовок страницы действие редактирования (EditAction), позволяя пользователю перейти к изменению текущей записи. Таким образом, класс обеспечивает базовый функционал для просмотра и редактирования категорий типов (см. листинг 70).

Листинг 70 – ViewTypeCategory controller.

class ViewTypeCategory extends ViewRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = TypeCategoryResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\EditAction::*make*(), *// Действие "Редактировать"* ];  
 }  
}

Контроллеры CreateUser, EditUser, ListUser, ViewUser

Класс CreateUser наследуется от CreateRecord и является контроллером страницы создания новой записи для ресурса UserResource. Связь с соответствующим ресурсом устанавливается через статическое свойство $resource. Класс предоставляет базовый функционал для создания новых пользователей в системе (см. листинг 71).

Листинг 71 – CreateUser controller.

class CreateUser extends CreateRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = UserResource::class;  
  
}

Класс EditUser наследуется от EditRecord и реализует контроллер для страницы редактирования записей ресурса UserResource. Связь с соответствующим ресурсом устанавливается через статическое свойство $resource. В заголовок страницы добавляются дополнительные действия для просмотра (ViewAction) и удаления (DeleteAction) текущей записи через метод getHeaderActions.

Метод save обеспечивает обработку сохранения данных формы: сначала получает состояние формы, затем обновляет и сохраняет модель записи. При необходимости выполняется перенаправление на страницу списка пользователей. Класс предоставляет комплексное решение для управления пользовательскими данными в системе (см. листинг 72).

Листинг 72 – EditUser controller.

class EditUser extends EditRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = UserResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\ViewAction::*make*(), *// Действие "Просмотр"* Actions\DeleteAction::*make*(), *// Действие "Удаление"* ];  
 }  
  
 *// Метод сохранения записи.* public function save(bool $shouldRedirect = true, bool $shouldSendSavedNotification = true): void  
 {  
 $data = $this->form->getState();  
  
 *// Примените изменения к данным* $this->record->update($data);  
  
 *// Сохраните модель, если нужно* $this->record->save();  
  
 *// Опционально: выполните дополнительные действия после сохранения* if ($shouldRedirect) {  
 *// Перенаправление пользователя после сохранения* $this->redirect($this->getResource()::*getUrl*('index'));  
 }  
 }  
}

Класс ListUsers наследуется от ListRecords и представляет контроллер страницы списка записей ресурса UserResource. Связь с ресурсом устанавливается через статическое свойство $resource.

Метод getHeaderActions добавляет в заголовок страницы действие создания новой записи (CreateAction), позволяя пользователю перейти к форме добавления нового пользователя. Таким образом, класс предоставляет базовый функционал для просмотра списка пользователей и создания новых записей (см. листинг 73).

Листинг 73 – ListUser controller.

class ListUsers extends ListRecords  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = UserResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\CreateAction::*make*(), *// Действие "Создать"* ];  
 }  
}

Класс ViewUser наследуется от ViewRecord и реализует контроллер для страницы просмотра детальной информации о записи ресурса UserResource. Связь с ресурсом устанавливается через статическое свойство $resource, что обеспечивает корректное отображение данных пользователя.

В функционал класса входит метод getHeaderActions, который добавляет в интерфейс страницы действие редактирования (EditAction). Это позволяет администраторам системы быстро переходить к изменению данных просматриваемого пользователя без необходимости возвращаться в общий список.

Таким образом, контроллер ViewUser предоставляет стандартный интерфейс для детального просмотра информации о пользователе с возможностью её последующего редактирования (см. листинг 74).

Листинг 74 – ViewUser controller.

class ViewUser extends ViewRecord  
{  
 *// Указание ресурса, с которым связана страница.* protected static string *$resource* = UserResource::class;  
  
 *// Получение дополнительных действий для заголовка страницы.* protected function getHeaderActions(): array  
 {  
 return [  
 Actions\EditAction::*make*(), *// Действие "Редактировать"* ];  
 }  
}

***Вывод изображений в админ-панель***

Для того, чтобы загружаемые изображения были видны в админ-панели, был прописан диск в Ресурсах Product и User

Класс UserResource наследуется от JsonResource и реализует преобразование данных пользователя в массив для API-ответа. Основная логика обработки данных содержится в методе toArray, который формирует структуру выходных данных.

В возвращаемом массиве передаются основные поля пользователя: имя (name), электронная почта (email), логин (login), пароль (password), идентификатор роли (role\_id). Особенностью реализации является обработка поля изображения (image), где используется метод Storage::disk("public")->url() для генерации полного URL-адреса изображения, хранящегося в публичном дисковом пространстве.

Таким образом, класс обеспечивает корректное формирование JSON-представления пользовательских данных, включая преобразование пути к изображению в доступный URL (см. листинг 75).

Листинг 75 – вывод изображения в UserResourse.

class UserResource extends JsonResource  
{  
 */\*\*  
 \* @return array<string, mixed>  
 \*/  
  
 //Преобразует ресурс в массив.* public function toArray(Request $request): array  
 {  
 return [  
 'name' => $this->name,  
 'email' => $this->email,  
 'login' => $this->login,  
 'password' => $this->password,  
 'image' => Storage::disk("public")->url($this->image),  
 'role\_id' => $this->role\_id,  
 ];  
 }  
}

Класс ProductResource преобразует данные продукта в массив через метод toArray. В результат включаются основные поля (id, title, price и др.), пути к изображениям через getImagesPath(), а также связанные ресурсы категории и заказов. Особенность - обработка связей с помощью TypeCategoryResource для категории и OrderResource::collection() для заказов (см. листинг 76).

Листинг 76 – вывод изображения в ProductResourse.

class ProductResource extends BaseJsonResource  
{  
 *//Преобразует ресурс в массив.* public function toArray(Request $request): array  
 {  
 return [  
 'id' => $this->id,  
 'download\_free' => $this->download\_free,  
 'title' => $this->title,  
 'comment' => $this->comment,  
 'price' => $this->price,  
 'year' => $this->year,  
 'images' => $this->getImagesPath(),  
 'type\_category\_id' => new TypeCategoryResource($this->whenLoaded('category')),  
 'orders' => OrderResource::*collection*($this->whenLoaded('orders')),  
 ];  
 }  
}

## **Маршрутизация**

Все маршруты Laravel должны быть определены в файлах маршрутов, находящихся в вашем каталоге routes. Эти файлы автоматически загружаются вашим поставщиком App\Providers\RouteServiceProvider приложения. Файл routes/web.php определяет маршруты для вашего веб-интерфейса. Этим маршрутам назначается группа посредников web, которая обеспечивает такие функции, как состояние сессии и защита от CSRF. Маршруты в routes/api.php не сохраняют состояния и им назначается группа посредников API.

Маршруты, определенные в файле routes/api.php, вложены в группу маршрутов в RouteServiceProvider. Внутри этой группы автоматически применяется префикс URL/api, поэтому вам не нужно вручную добавлять его к каждому маршруту в файле маршрутов. Вы можете изменить префикс и другие параметры группы маршрутов, изменив свой класс RouteServiceProvider.

Исходя из технического задания были разработаны следующие маршруты, которые записаны в файле routes\api.php (см. листинг 77):

Листинг 77 – api маршруты.

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {  
 return $request->user();  
});  
  
Route::group(['middleware' => 'auth:sanctum'], function () {  
 *//Route::get('/get', 'GetController');*});  
  
*//Тип категории продукта*Route::controller(TypeCategoryController::class)->group(function () {  
 Route::get('/type-category', 'index');  
});  
  
*//Продукты*Route::controller(ProductController::class)->group(function () {  
 Route::get('/product', 'index');  
 Route::get('/product/category/{type\_category\_id}', 'filter');  
 Route::get('/products/filter', 'filterByPrice');  
});  
  
*//Заказы*Route::controller(OrderController::class)->group(function () {  
 Route::get('/order', 'index')->middleware('auth:sanctum');  
});  
  
*//Корзина*Route::controller(CartController::class)->group(function () {  
 Route::get('/cart', 'index')->middleware('auth:sanctum');  
 Route::post('/cart/add-product/{id}', 'addItem')->middleware('auth:sanctum');  
 Route::delete('/cart/delete-product/{id}', 'removeItem')->middleware('auth:sanctum');  
});  
  
*//API-TOKEN*Route::post('/personal-access-tokens', [PersonalAccessToken::class, 'store'])->middleware('auth:sanctum');  
Route::delete('/personal-access-tokens/{tokenId}', [PersonalAccessToken::class, 'delete'])->middleware('auth:sanctum');  
  
  
*//Регистрация*Route::post('/register', [AuthUserController::class, 'store'])->middleware('guest');  
  
*//Авторизация*Route::post('/login', [AuthUserController::class, 'login'])->name('login');  
  
*//Деавторизация*Route::middleware('auth:sanctum')->delete('/logout', [AuthUserController::class, 'logout']);  
*//Поисковая строка*Route::get('/search', [SearchController::class, 'search'])->name('search');

Маршруты, которые записаны в файле routes\web.php (см. листинг 78):

Листинг 78 – web маршруты.

*//Регистрация*Route::post('/register', [AuthUserController::class, 'store'])->middleware('guest');  
  
*//Авторизация*Route::post('/login', [AuthUserController::class, 'login']);  
  
*//Деавторизация*Route::delete('/logout', [AuthUserController::class, 'logout'])->middleware('auth');  
  
*//Поисковая строка*Route::get('/search', [SearchController::class, 'search'])->name('search');

# **ГЛАВА 7. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ**

Для работы с тестированием API приложения нам потребуется программа Postman.

**Аутентификация.**

Если клиент отправил корректные данные, то возвращается сгенерированный токен (см. рисунок 50 и листинг 79), в противном случае сообщение об ошибке авторизации (см. рисунок 51).

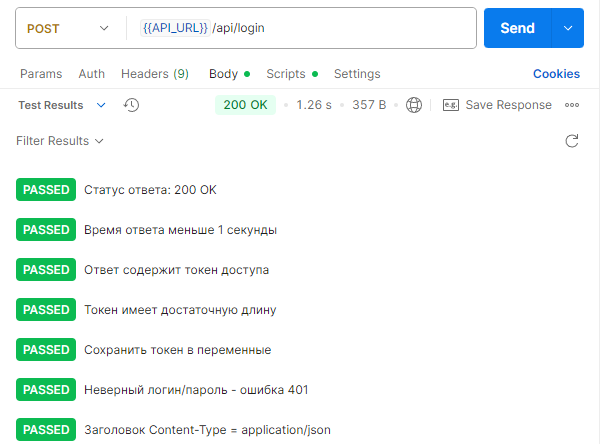


Рисунок 50 – Результат теста

Листинг 79 – авто-тест на авторизацию.

// 1. Проверка успешного статуса ответа

pm.**test**("Статус ответа: 200 OK", **function**() {

    pm.response.to.have.status(200);

});

// 2. Проверка времени ответа

pm.**test**("Время ответа меньше 1 секунды", **function**() {

    pm.expect(pm.response.responseTime).to.be.below(2000);

});

// 3. Проверка структуры ответа

pm.**test**("Ответ содержит токен доступа", **function**() {

    const response **=** pm.response.json();

    pm.expect(response).to.be.an('object');

    pm.expect(response).to.have.all.**keys**('access\_token', 'token\_type');

    pm.expect(response.access\_token).to.be.a('string').and.to.not.be.empty;

    pm.expect(response.token\_type).to.eql('Bearer');

});

// 4. Проверка длины токена

pm.**test**("Токен имеет достаточную длину", **function**() {

    const token **=** pm.response.json().access\_token;

    pm.expect(token.length).to.be.above(30);

});

// 5. Сохранение токена для последующих запросов

pm.**test**("Сохранить токен в переменные", **function**() {

    const response **=** pm.response.json();

    pm.environment.**set**('auth\_token', response.access\_token);

    pm.expect(pm.environment.**get**('auth\_token')).to.not.be.empty;

});

// 6. Негативный тест: неверные учетные данные

pm.**test**("Неверный логин/пароль - ошибка 401", **async** () **=>** {

    const invalidAuthRequest **=** {

        url: pm.request.url.**toString**(),

        method: "POST",

        body: {

            mode: 'urlencoded',

            urlencoded: [

                { key: 'login', value: 'wrong' },

                { key: 'password', value: 'credentials' }

            ]

        }

    };

    const response **=** **await** pm.sendRequest(invalidAuthRequest);

    pm.expect(response.code).to.equal(401);

});

// 7. Проверка заголовков

pm.**test**("Заголовок Content-Type = application/json", **function**() {

    pm.expect(pm.response.headers.**get**('Content-Type')).to.include('application/json');

});

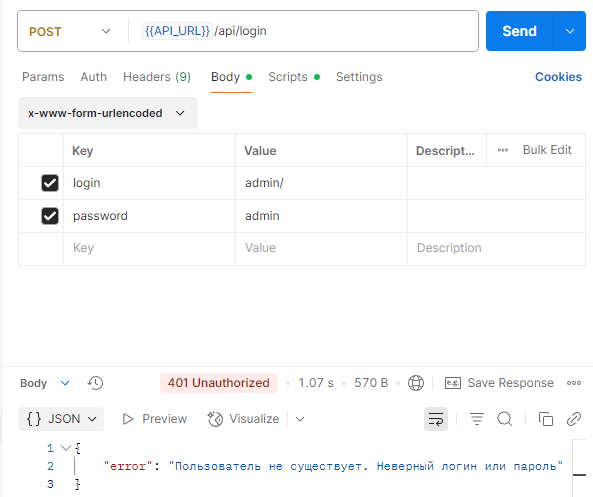


Рисунок 51 – Ошибка авторизации

**Выход.**

Запрос предназначен для очистки значение токена пользователя, другими словами, выход из учетной записи (см. рисунок 52,53 и листинг 80).

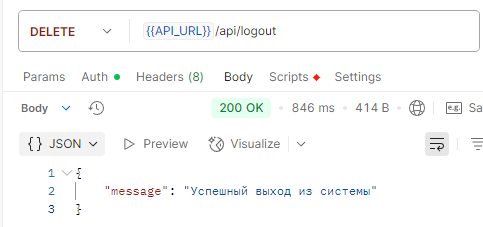


Рисунок 52 – Выход из учетной записи

Листинг 80 – авто-тест на деавторизацию.

// 1. Тест успешного выхода

pm.**test**("Успешный выход - статус 200", **function**() {

    pm.expect(pm.response.code).to.equal(200);

    pm.expect(pm.response.json().message).to.include('выход');

});

// 2. Проверка недействительности токена (с обработкой ошибок)

pm.**test**("Токен недействителен после выхода", **function**() {

**return** **new** Promise((resolve, reject) **=>** {

        const testRequest **=** {

            url: pm.environment.**get**("base\_url") **+** "/api/user",

            method: "GET",

            header: {

                'Authorization': 'Bearer ' **+** pm.environment.**get**("auth\_token"),

                'Accept': 'application/json'

            }

        };

        pm.sendRequest(testRequest, **function**(err, res) {

**if** (err) **return** reject(err);

**try** {

                pm.expect(res.code).to.be.oneOf([401, 404, 403]);

                resolve();

            } **catch** (assertErr) {

                reject(assertErr);

            }

        });

    });

});

// 3. Альтернативный тест с явным catch

pm.**test**("Проверка защищенного маршрута", **function**() {

**return** pm.sendRequest({

        url: pm.environment.**get**("base\_url") **+** "/api/protected",

        method: "GET",

        header: {

            'Authorization': 'Bearer ' **+** pm.environment.**get**("auth\_token")

        }

    })

    .**then**(res **=>** {

        pm.expect(res.code).to.equal(401);

    })

    .**catch**(err **=>** {

        pm.expect.fail("Ошибка при проверке: " **+** err.message);

    });

});

// 4. Проверка с невалидным токеном

pm.**test**("Невалидный токен - ошибка 401", **function**() {

**return** pm.sendRequest({

        url: pm.environment.**get**("base\_url") **+** "/api/user",

        method: "GET",

        header: {

            'Authorization': 'Bearer невалидный\_токен'

        }

    })

    .**then**(res **=>** {

        pm.expect(res.code).to.equal(401);

    })

    .**catch**(() **=>** {

        // Игнорируем ошибки соединения

    });

});

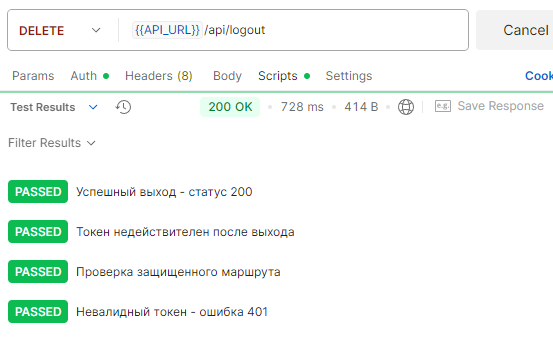


Рисунок 53 - Результат теста

**Регистрация.**

Регистрация пользователя. При попытке данных, не проходящих валидацию, будут выводится ошибки. (см. рисунок 54, 55) и Листинг 81.

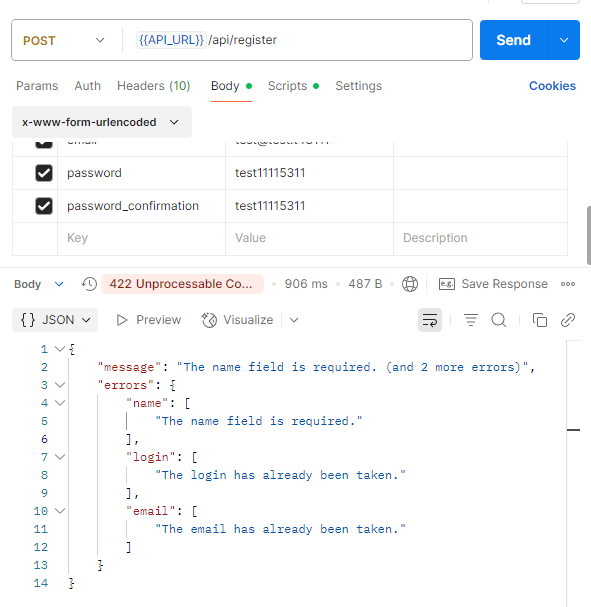


Рисунок 54 – Валидация регистрации

Листинг 81 – авто-тест на регистрацию.

// 1. Проверка успешной регистрации (201 Created)

pm.**test**("Успешная регистрация - статус 201", **function**() {

    pm.response.to.have.status(201);

});

// 2. Проверка структуры ответа

pm.**test**("Ответ содержит данные пользователя", **function**() {

    const response **=** pm.response.json();

    pm.expect(response).to.be.an('object');

    pm.expect(response).to.have.property('user');

    pm.expect(response.user).to.include.**keys**('id', 'name', 'email', 'login');

});

// 3. Проверка времени ответа

pm.**test**("Время ответа меньше 1.5 секунды", **function**() {

    pm.expect(pm.response.responseTime).to.be.below(1500);

});

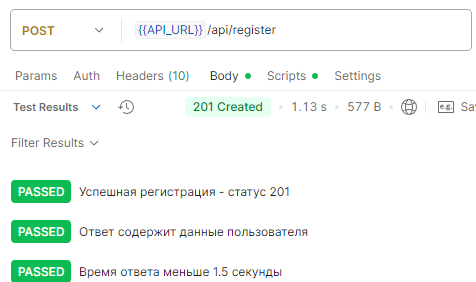


Рисунок 55 - Результат теста

## **Тесты функционала администратора**

Так как весь функционал выполняется в админ-панели, а не в программе Postman, то при неверно заполненных данных или пустых полях, выдается ошибка, которую можно наблюдать на рисунке 56.

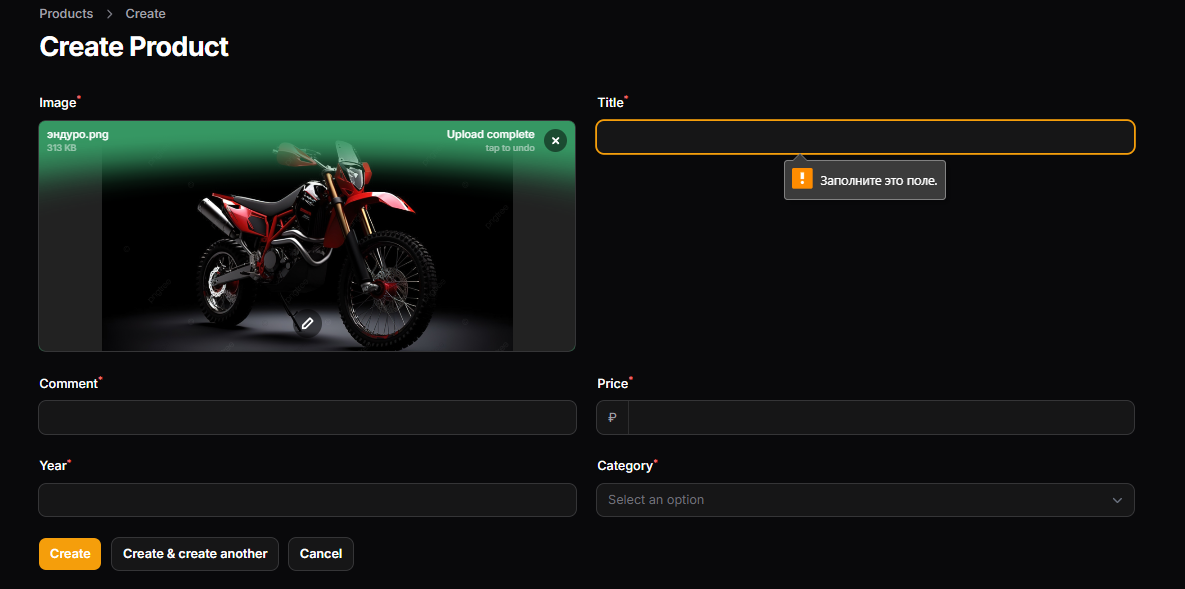


Рисунок 56 – Валидация данных

## **Тесты функционала пользователя**

**Добавление товара в корзину.**

Авторизированные пользователь может добавить любой товар в корзину в любом кол-ве, также просмотреть ее и удалить товар из корзины (см. рисунок 57-63) и Листинг 82-84.

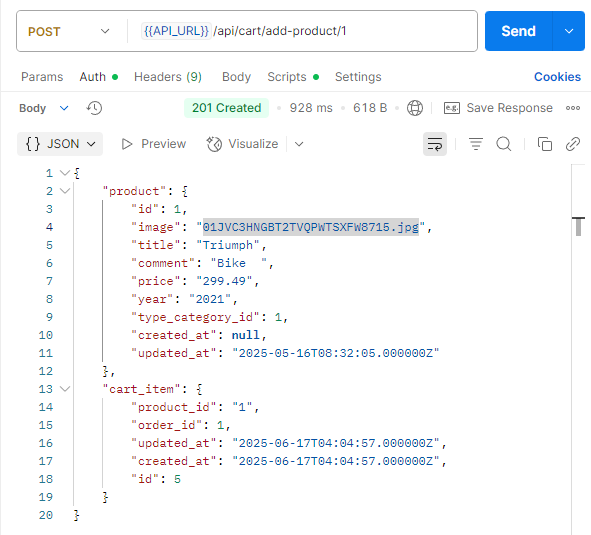


Рисунок 57 – Добавление в корзину

Листинг 82 – авто-тест на добавление в корзину.

// 1. Проверка статуса ответа

pm.**test**("Статус 201 Created", **function**() {

    pm.response.to.have.status(201);

});

// 2. Проверка времени ответа

pm.**test**("Время ответа меньше 1000мс", **function**() {

    pm.expect(pm.response.responseTime).to.be.below(1000);

});

// 3. Проверка структуры ответа

pm.**test**("Ответ содержит объект product и cart\_item", **function**() {

    const response **=** pm.response.json();

    pm.expect(response).to.be.an('object');

    pm.expect(response).to.have.**keys**('product', 'cart\_item');

});

// 4. Проверка данных товара

pm.**test**("Информация о товаре", **function**() {

    const product **=** pm.response.json().product;

    pm.expect(product).to.include.**keys**('id', 'image', 'title', 'price', 'year', 'type\_category\_id');

    pm.expect(product.id).to.be.a('number');

    pm.expect(product.title).to.be.a('string').and.to.not.be.empty;

    pm.expect(product.price).to.**match**(/**^**\d**+**\.\d**{2}$**/);

    pm.expect(product.year).to.**match**(/**^**\d**{4}$**/);

});

// 5. Проверка элемента корзины

pm.**test**("Информация о cart\_item", **function**() {

    const cartItem **=** pm.response.json().cart\_item;

    pm.expect(cartItem).to.include.**keys**('product\_id', 'order\_id', 'id');

    pm.expect(cartItem.product\_id).to.eql("1");

    pm.expect(cartItem.order\_id).to.be.a('number');

    pm.expect(cartItem.id).to.be.a('number');

});

// 6. Проверка дат создания/обновления

pm.**test**("Даты созданы и корректны", **function**() {

    const cartItem **=** pm.response.json().cart\_item;

    const product **=** pm.response.json().product;

**if** (cartItem.created\_at) {

        pm.expect(**new** Date(cartItem.created\_at)).to.be.a('date');

    }

**if** (cartItem.updated\_at) {

        pm.expect(**new** Date(cartItem.updated\_at)).to.be.a('date');

    }

**if** (product.updated\_at) {

        pm.expect(**new** Date(product.updated\_at)).to.be.a('date');

    }

});

// 7. Дополнительная проверка изображения

pm.**test**("Изображение товара валидно", **function**() {

    const image **=** pm.response.json().product.image;

    pm.expect(image).to.be.a('string');

    pm.expect(image).to.**match**(/\.(jpg**|**jpeg**|**png)**$**/**i**);

});

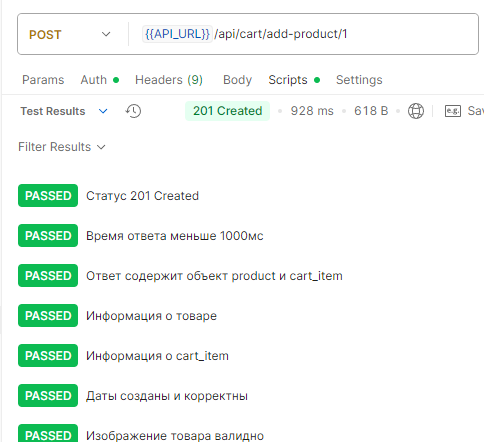


Рисунок 58 Результат теста

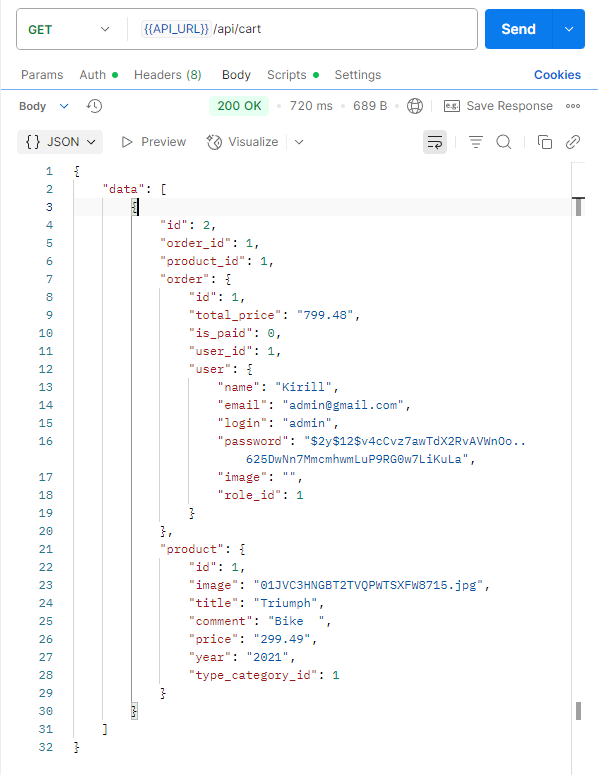


Рисунок 59 – Просмотр корзины

Листинг 83 – авто-тест на просмотр корзины.

// 1. Проверка статуса ответа

pm.**test**("Статус ответа: 200 OK", **function** () {

    pm.response.to.have.status(200);

});

// 2. Проверка времени ответа

pm.**test**("Время ответа меньше 1000мс", **function** () {

    pm.expect(pm.response.responseTime).to.be.below(1000);

});

// 3. Проверка структуры основного ответа

pm.**test**("Основная структура ответа", **function**() {

    const response **=** pm.response.json();

    pm.expect(response).to.be.an('object');

    pm.expect(response).to.have.property('data').that.is.an('array');

    pm.expect(response.data[0]).to.include.**keys**('id', 'order\_id', 'product\_id', 'order');

});

// 4. Проверка структуры заказа

pm.**test**("Структура заказа", **function** () {

    const response **=** pm.response.json();

    const order **=** response.data[0].order;

    pm.expect(order).to.be.an('object').that.is.not.undefined.and.is.not.null;

    pm.expect(order).to.include.**keys**('id', 'total\_price');

    pm.expect(order.total\_price).to.**match**(/**^**\d**+**\.\d**{1,2}**[P]**?$**/);

    // Опциональные поля

**if** (order.hasOwnProperty('is\_paid')) {

        pm.expect(order.is\_paid).to.be.a('number');

    }

});

// 5. Проверка товара в корзине

pm.**test**("Информация о товаре", **function** () {

    const product **=** pm.response.json().data[0].product;

    pm.expect(product).to.have.**keys**('id', 'image', 'title', 'comment', 'price', 'year', 'type\_category\_id');

    pm.expect(product.price).to.**match**(/**^**\d**+**\.\d**{1,2}**[P]**?$**/);

    pm.expect(product.year).to.**match**(/**^**\d**{4}$**/);

});

// 6. Проверка формата пароля (базовая)

pm.**test**("Пароль зашифрован", **function** () {

    const password **=** pm.response.json().data[0].order.user.password;

    pm.expect(password).to.be.a('string').and.to.have.lengthOf.above(20);

});

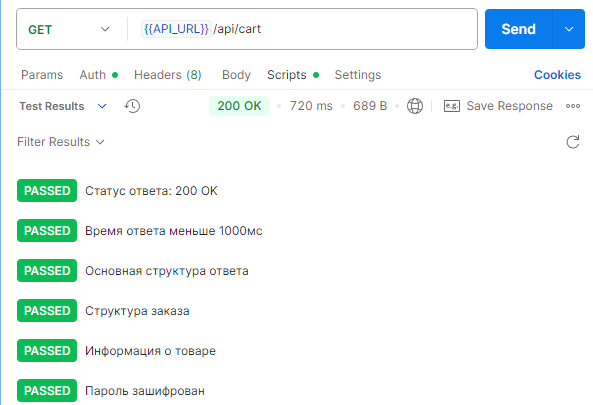


Рисунок 60 Результат теста

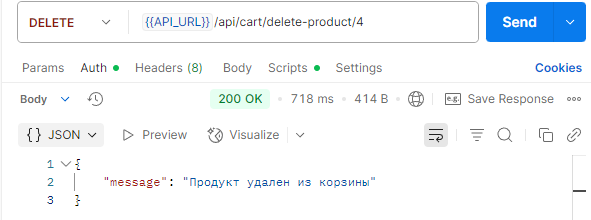


Рисунок 61 – Удаление товара из корзины

Листинг 84 – авто-тест удаление товара из корзины.

// 1. Проверка статуса ответа

pm.**test**("Статус 200 OK", **function**() {

    pm.response.to.have.status(200);

});

// 2. Проверка времени ответа

pm.**test**("Время ответа меньше 1000мс", **function**() {

    pm.expect(pm.response.responseTime).to.be.below(1000);

});

// 3. Проверка формата ответа

pm.**test**("Ответ содержит сообщение об успешном удалении", **function**() {

    const response **=** pm.response.json();

    pm.expect(response).to.be.an('object');

    pm.expect(response).to.have.property('message');

    pm.expect(response.message).to.be.a('string').and.to.not.be.empty;

});

// 4. Проверка текста сообщения

pm.**test**("Корректное сообщение об удалении", **function**() {

    const response **=** pm.response.json();

    pm.expect(response.message).to.**match**(/продукт.**\***удален/**i**);

    // Или точное совпадение, если известно:

    // pm.expect(response.message).to.eql("Продукт удален из корзины");

});

// 5. Проверка авторизации (необязательно)

pm.**test**("Требуется авторизация", **function**() {

    const authHeader **=** pm.request.headers.**get**('Authorization');

    pm.expect(authHeader).to.exist.and.to.include('Bearer');

});

// 6. Негативный тест для несуществующего товара

pm.**test**("Несуществующий товар - ошибка 404", **async** () **=>** {

    const invalidRequest **=** {

        url: pm.request.url.**toString**().**replace**(/delete-product\/\d**+**/, 'delete-product/9999'),

        method: "DELETE",

        header: pm.request.headers.toObject()

    };

    const response **=** **await** pm.sendRequest(invalidRequest);

    pm.expect(response.code).to.equal(404);

});

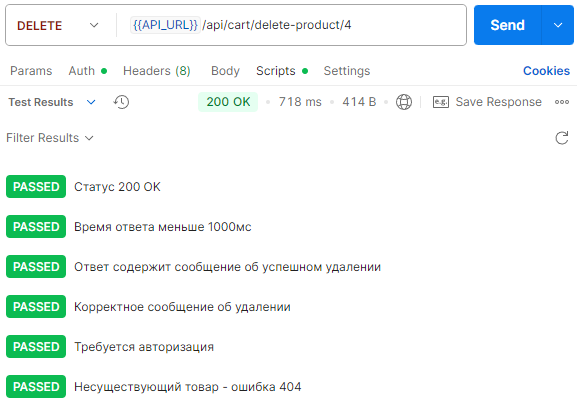


Рисунок 62 Результат теста

Ниже можно наблюдать проверку наличия товара в корзине.

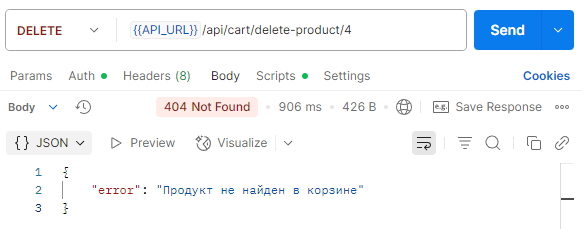


Рисунок 63 – Ошибка удаление

**Поисковая строка.**

В поисковую строку может вписать любую букву и получить подходящий товар, как авторизированный пользователь, так и гость (см. рисунок 64, 65) и Листинг 85.

****

Рисунок 64 Результат на буквы Tr

Листинг 85 – авто-тест на получение товаров используя поисковую строку.

// 1. Проверка статуса ответа

pm.**test**("Статус ответа 200 OK", **function**() {

    pm.response.to.have.status(200);

});

// 2. Проверка структуры найденных продуктов

pm.**test**("Продукты имеют правильную структуру", **function**() {

    const products **=** pm.response.json().products;

**if** (products.length **>** 0) {

        const firstProduct **=** products[0];

        pm.expect(firstProduct).to.include.**keys**(

            'id', 'image', 'title', 'price', 'year', 'type\_category\_id'

        );

        // Проверка типов данных

        pm.expect(firstProduct.id).to.be.a('number');

        pm.expect(firstProduct.title).to.be.a('string').and.to.not.be.empty;

        pm.expect(firstProduct.price).to.**match**(/**^**\d**+**\.\d**{2}$**/);

    }

});

// 3. Проверка времени ответа

pm.**test**("Время ответа меньше 1 секунды", **function**() {

    pm.expect(pm.response.responseTime).to.be.below(1000);

});

// 4. Проверка пустого запроса

pm.**test**("Пустой поисковый запрос возвращает массив", **async** () **=>** {

    const emptySearch **=** **await** pm.sendRequest({

        url: pm.request.url.**toString**().**replace**('?search=T', ''),

        method: "GET"

    });

    pm.expect(emptySearch.code).to.equal(200);

    pm.expect(emptySearch.json().products).to.be.an('array');

});

// 5. Проверка чувствительности к регистру

pm.**test**("Поиск чувствителен к регистру", **async** () **=>** {

    const lowerCaseSearch **=** **await** pm.sendRequest({

        url: pm.request.url.**toString**().**replace**('?search=T', '?search=t'),

        method: "GET"

    });

    // Ожидаем либо другой результат, либо пустой массив

    pm.expect(lowerCaseSearch.json().products).to.be.an('array');

});

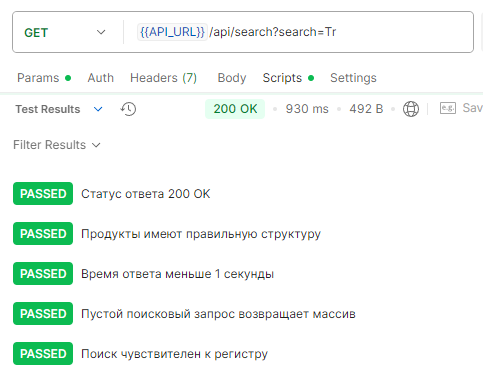
****

Рисунок 65 Результат теста

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения дипломного проекта мной было проанализировано техническое задание на разработку MotoShop, был разработан пользовательский интерфейс и реализован функционал сервиса API.

Дальнейшие глобальные цели и задачи проекта написаны на 6 странице отчета.

В ходе работы мной были улучшены навыки веб-разработки, которые пригодятся мне в моей дальнейшей профессиональной деятельности. Цель дипломного проекта была полностью достигнута, а все задачи выполнены, а именно:

1. Произведен анализ предметной области.
2. Спроектировано техническое задание.
3. ВСоздан адаптивный макет будущего сайта.
4. Спроектирована и разработана база данных.
5. Спроектирован и разработан функционал информационного продукта.
6. Создан продающий продукт.
7. Написан пользовательский интерфейс
8. Разработан функционал сайта на стороне клиента.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Обучающий центр Postman - https://learning.postman.com/
2. Сообщество ИТ специалистов Хабр - https://habr.com/ru/feed/
3. Видео хостинг YouTube - https://www.youtube.com/
4. Vue3 руководство - <https://v3.ru.vuejs.org/ru/guide/introduction.html>
5. Официальная документация Laravel - https://laravel.com/docs/11.x
6. Обучающий центр Postman - https://learning.postman.com/
7. Руководство по PHP - https://www.php.net/manual/ru/
8. Документация Open Server Panel - https://ospanel.io/docs/
9. Сообщество ИТ специалистов Хабр - https://habr.com/ru/feed/
10. Видео хостинг YouTube - https://www.youtube.com/
11. Обучающие статьи по Laravel - https://laravel.demiart.ru/
12. Официальная документация Filament - https://filamentphp.com/
13. Учебник по программированию code.mu - https://code.mu/ru/