

Реализация системы управления товарами (CRUD) в интернет-магазине*Ран Максим Александрович**Приамурский государственный университет имени Шолом-Алейхема**Студент***Аннотация**

В статье рассматривается реализация системы управления товарным каталогом в веб-приложении интернет-магазина. Основное внимание уделено организации CRUD-операций для различных типов товаров, включая мотоциклы, снегоходы и запчасти, а также взаимодействию административной панели с серверной частью и базой данных. Описывается архитектура решения на основе Node.js, Express и ORM Prisma, обеспечивающая централизованное управление данными, контроль доступа и целостность информации. Показано, что использование ролевой модели и REST API позволяет эффективно разграничить функциональность пользовательской и административной частей системы, а также упростить расширение и сопровождение приложения.

Ключевые слова: интернет-магазин, управление товарами, CRUD, административная панель, REST API, Prisma, Node.js, React.

Implementation of a product management system (CRUD) in an online store*Ran Maksim Aleksandrovich**Sholom-Aleichem Priamursky State University**Student***Abstract**

The article discusses the implementation of a product catalog management system in the web application of an online store. The main focus is on the organization of CRUD operations for various types of goods, including motorcycles, snowmobiles and spare parts, as well as the interaction of the administrative panel with the server and database. The architecture of the Node-based solution is described.js, Express, and ORM Prisma, which provides centralized data management, access control, and information integrity. It is shown that using the role model and the REST API makes it possible to effectively differentiate the functionality of the user and administrative parts of the system, as well as simplify the expansion and maintenance of the application.

Keywords: online store, product management, CRUD, administrative panel, REST API, Prisma, Node.js, React.

1 Введение

1.1 Актуальность

Современные интернет-магазины предъявляют высокие требования к управлению товарным каталогом, который должен оперативно обновляться, масштабироваться и поддерживать различные типы продукции. Для торговых платформ, ориентированных на продажу специализированной техники, таких как мототехника и комплектующие, особое значение имеет удобство администрирования, корректное хранение характеристик товаров и надежность операций добавления, редактирования и удаления данных. В этих условиях актуальной задачей является разработка системы управления товарами, которая обеспечивает единый подход к работе с разнородными сущностями, поддерживает загрузку мультимедийных данных и реализует разграничение прав доступа. Использование современных веб-технологий и ORM-инструментов позволяет повысить надежность и сопровождаемость таких систем, что обуславливает актуальность рассматриваемой темы.

1.2 Обзор исследований

Разработка веб-приложений на основе React и TypeScript активно рассматривается в научной литературе. В работе М.М. Ковцура, В.Е.Филимонова и Е.К. Щёголева анализируются подходы к созданию многопользовательских систем на базе React, где особое внимание уделяется организации типизации и архитектуры клиентской части [1]. А.А. Дудак в своём исследовании проводит сравнительный анализ инструментов разработки для проектных систем, выделяя преимущества стека технологий React и TypeScript по производительности и удобству сопровождения кода [2]. В статье Д.А. Хайруллина и М.Р. Богданова рассматриваются особенности применения статической типизации в React-приложениях, включая методики типизации компонентов, хуков и состояния, а также влияние строгой типизации на надёжность пользовательских интерфейсов [3]. К.Р. Удальцов исследует вопросы обеспечения качества кода в командной разработке, предлагая формализацию шаблонов безопасных pull request-ов для проектов на React и TypeScript в рамках Git Flow [4]. Особое внимание созданию библиотек компонентов на основе TypeScript уделяется в работе Т.М.Воронко, где описываются принципы модульности, повторного использования кода и проектирования архитектуры компонентов для масштабируемых React-решений [5].

1.3 Цель исследования

Целью данной работы является разработка и описание системы управления товарами (CRUD) в интернет-магазине, обеспечивающей централизованное администрирование товарного каталога через защищённый интерфейс. В рамках исследования ставится задача проанализировать архитектурные решения, применяемые для взаимодействия административной панели, серверной логики и базы данных, а также показать

практическую реализацию операций создания, чтения, обновления и удаления товаров с учетом ролевой авторизации и обработки связанных данных.

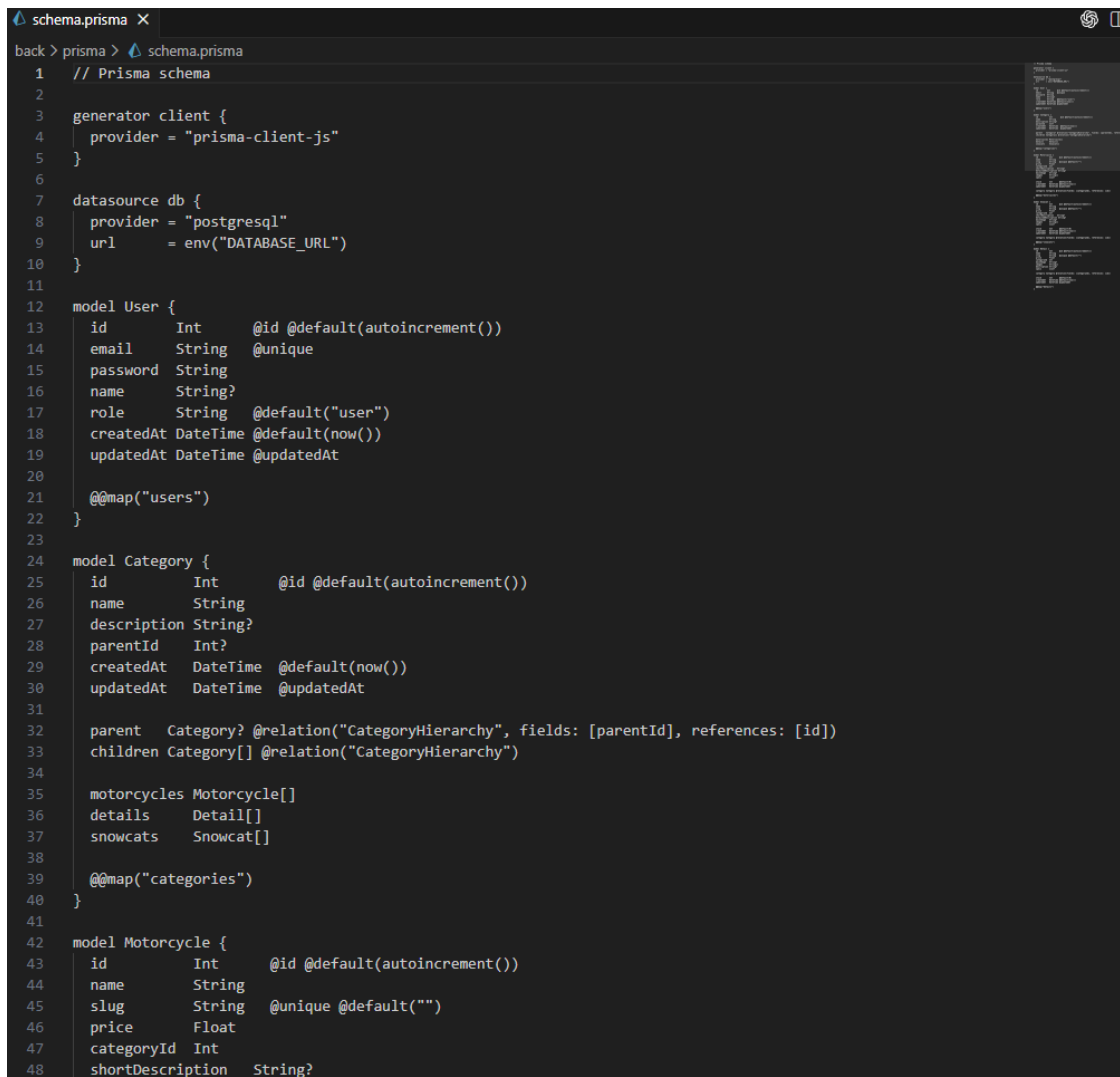
2 Материалы и методы

В ходе работы использовалось веб-приложение интернет-магазина, разработанное с применением клиентской части на React и TypeScript, серверной части на Node.js с использованием фреймворка Express, а также реляционной базы данных PostgreSQL. Для работы с базой данных применялась ORM Prisma, обеспечивающая типобезопасный доступ к данным и упрощение реализации CRUD-операций. Административный доступ к операциям управления товарами реализован с использованием JWT-аутентификации и серверных middleware для проверки прав пользователя. Взаимодействие между компонентами системы осуществляется через REST API, а обработка изображений и вспомогательных данных выполняется на стороне сервера.

3 Результаты и обсуждения

В результате выполненной разработки была реализована система управления товарами (CRUD), обеспечивающая централизованное администрирование товарного каталога интернет-магазина. Система охватывает полный цикл работы с данными о товарах, включая создание, чтение, обновление и удаление записей, а также обработку связанных сущностей, таких как категории, изображения и характеристики продукции.

Основой системы управления товарами является спроектированная структура базы данных, реализованная с использованием ORM Prisma. Для представления товаров применяются отдельные модели, соответствующие различным типам продукции: мотоциклы, снегоходы и запчасти. Каждая модель содержит уникальный идентификатор, символьный slug для формирования человекочитаемых URL, ценовые параметры, ссылки на изображения, JSON-поле для хранения технических характеристик и поле stock для учёта наличия товара. Фрагмент схемы базы данных, отражающий структуру товарных моделей и их связь с категориями, представлен на рисунке (рис. 1).



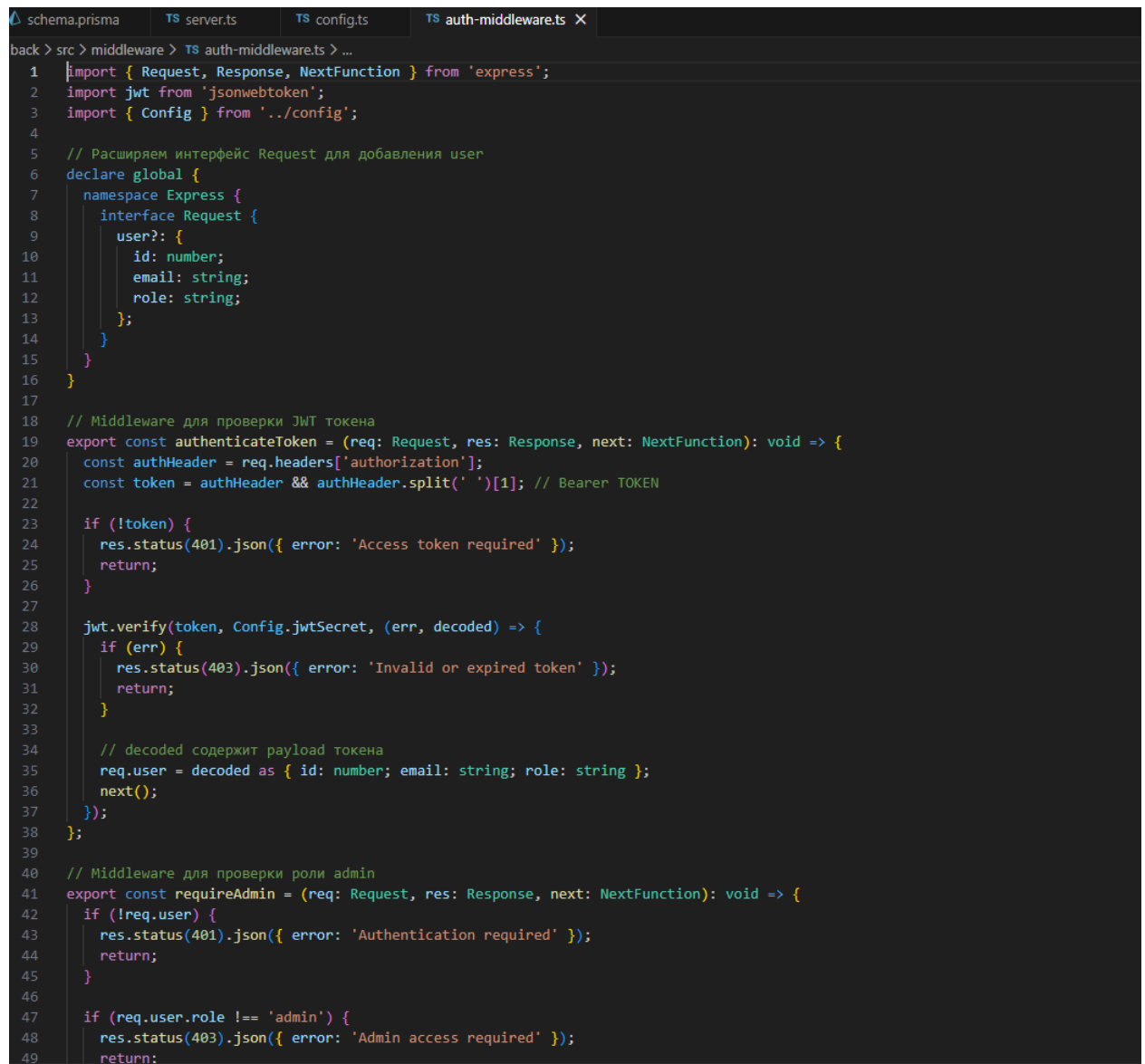
```

1 // Prisma schema
2
3 generator client {
4   provider = "prisma-client-js"
5 }
6
7 datasource db {
8   provider = "postgresql"
9   url      = env("DATABASE_URL")
10 }
11
12 model User {
13   id      Int      @id @default(autoincrement())
14   email   String   @unique
15   password String
16   name    String?
17   role    String   @default("user")
18   createdAt DateTime @default(now())
19   updatedAt DateTime @updatedAt
20
21   @@map("users")
22 }
23
24 model Category {
25   id      Int      @id @default(autoincrement())
26   name    String
27   description String?
28   parentId Int?
29   createdAt DateTime @default(now())
30   updatedAt DateTime @updatedAt
31
32   parent Category? @relation("CategoryHierarchy", fields: [parentId], references: [id])
33   children Category[] @relation("CategoryHierarchy")
34
35   motorcycles Motorcycle[]
36   details Detail[]
37   snowcats Snowcat[]
38
39   @@map("categories")
40 }
41
42 model Motorcycle {
43   id      Int      @id @default(autoincrement())
44   name    String
45   slug    String   @unique @default("")
46   price   Float
47   categoryId Int
48   shortDescription String?

```

Рис. 1. Модели товаров и категорий в схеме базы данных Prisma)

Для управления данными был реализован набор серверных маршрутов, доступных исключительно авторизованным пользователям с ролью администратора. Все административные маршруты защищены middleware-аутентификации, выполняющим проверку JWT-токена и прав доступа пользователя. Это решение позволяет исключить несанкционированное выполнение операций CRUD и обеспечивает разграничение функциональности между публичной и административной частями приложения. Пример применения middleware аутентификации и проверки роли администратора показан на рисунке (рис. 2).



```
back > src > middleware > TS auth-middleware.ts > ...
1  import { Request, Response, NextFunction } from 'express';
2  import jwt from 'jsonwebtoken';
3  import { Config } from '../config';
4
5  // Расширяем интерфейс Request для добавления user
6  declare global {
7    namespace Express {
8      interface Request {
9        user?: {
10          id: number;
11          email: string;
12          role: string;
13        };
14      }
15    }
16  }
17
18  // Middleware для проверки JWT токена
19  export const authenticateToken = (req: Request, res: Response, next: NextFunction): void => {
20    const authHeader = req.headers['authorization'];
21    const token = authHeader && authHeader.split(' ')[1]; // Bearer TOKEN
22
23    if (!token) {
24      res.status(401).json({ error: 'Access token required' });
25      return;
26    }
27
28    jwt.verify(token, Config.jwtSecret, (err, decoded) => {
29      if (err) {
30        res.status(403).json({ error: 'Invalid or expired token' });
31        return;
32      }
33
34      // decoded содержит payload токена
35      req.user = decoded as { id: number; email: string; role: string };
36      next();
37    });
38  };
39
40  // Middleware для проверки роли admin
41  export const requireAdmin = (req: Request, res: Response, next: NextFunction): void => {
42    if (!req.user) {
43      res.status(401).json({ error: 'Authentication required' });
44      return;
45    }
46
47    if (req.user.role !== 'admin') {
48      res.status(403).json({ error: 'Admin access required' });
49      return;
50    }
51    next();
52  };
53  }
```

Рис. 2. Middleware проверки JWT-токена и роли администратора

Операция создания товара реализована через административные маршруты, принимающие данные формы, загружаемые изображения и вспомогательные параметры. При добавлении нового товара автоматически формируется slug на основе названия, что обеспечивает уникальность и удобство последующей навигации. Загрузка изображений осуществляется на стороне сервера с использованием специализированного middleware, после чего пути к файлам сохраняются в базе данных. Код обработки создания товара и загрузки изображений представлен на рисунке (рис. 3).

```
back > src > routes > TS admin.ts > ...
1  import { Router } from 'express';
2  import { PrismaClient } from '@prisma/client';
3  import { authenticateToken, requireAdmin } from '../middleware/auth-middleware';
4  import multer from 'multer';
5  import path from 'path';
6  import fs from 'fs';
7  import { promisify } from 'util';
8  import slugify from 'slugify';
9
10 const router = Router();
11 const prisma = new PrismaClient();
12
13 // Настройка multer для загрузки изображений
14 const storage = multer.diskStorage({
15   destination: (req, file, cb) => {
16     cb(null, path.join(__dirname, '../..../uploads'));
17   },
18   filename: (req, file, cb) => {
19     const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
20     cb(null, uniqueSuffix + path.extname(file.originalname));
21   }
22 });
23
24 const upload = multer({
25   storage,
26   limits: {
27     fileSize: 5 * 1024 * 1024, // 5MB
28   },
29   fileFilter: (req, file, cb) => {
30     const allowedTypes = /jpeg|jpg|png|gif/;
31     const extname = allowedTypes.test(path.extname(file.originalname).toLowerCase());
32     const mimetype = allowedTypes.test(file.mimetype);
33
34     if (mimetype && extname) {
35       return cb(null, true);
36     } else {
37       cb(new Error('Только изображения разрешены!'));
38     }
39   }
40 });
41
42 // Все админские маршруты требуют аутентификации и роли admin
43 router.use(authenticateToken);
44 router.use(requireAdmin);
45
```

Рис. 3. Реализация операции создания товара и обработки загружаемых изображений

Чтение данных о товарах реализовано как через административную панель, так и через публичный API. Администратор получает расширенную информацию о товарах, включая наличие, категории и дополнительные характеристики, тогда как пользовательская часть использует упрощённые GET-запросы для отображения каталога и детальных страниц товаров. Для получения информации о конкретном товаре применяется параметрическая маршрутизация по slug, что позволяет использовать единый URL-формат независимо от типа товара. Пример серверной логики поиска товара по slug представлен на рисунке (рис. 4).

```
93 // Получение товара по slug
94 app.get('/api/products/:slug', async (req, res) => {
95   try {
96     const { slug } = req.params;
97
98     console.log('Searching for product with slug:', slug);
99
100    // Ищем товар сначала среди мотоциклов, затем среди запчастей, затем среди снегоходов
101    let product: any = await prisma.motorcycle.findFirst({
102      where: {
103        slug: slug
104      },
105      include: {
106        category: true
107      }
108    });
109
110    if (!product) {
111      product = await prisma.detail.findFirst({
112        where: {
113          slug: slug
114        },
115        include: {
116          category: true
117        }
118      });
119    }
120
121    if (!product) {
122      product = await prisma.snowcat.findFirst({
123        where: {
124          slug: slug
125        },
126        include: {
127          category: true
128        }
129      });
130    }
131
132    console.log('Found product:', product ? product.name : 'null');
133
134    if (!product) {
135      return res.status(404).json({ error: 'Товар не найден' });
```

Рис. 4. Получение товара по slug с последовательным поиском по типам продукции

Операция обновления товара включает изменение текстовых данных, характеристик, категорий и изображений. При обновлении изображений предусмотрен механизм удаления устаревших файлов с сервера, что предотвращает накопление неиспользуемых данных и снижает нагрузку на файловую систему. Данный подход повышает устойчивость и аккуратность хранения мультимедийных ресурсов. Фрагмент логики обновления товара с удалением старых изображений показан на рисунке (рис. 5).

```
132
133 // Проверяем основное изображение
134 if (currentMotorcycle.mainImage &&
135     currentMotorcycle.mainImage.startsWith('http://localhost:5000/uploads/') &&
136     mainImage !== currentMotorcycle.mainImage) {
137     imagesToDelete.push(currentMotorcycle.mainImage);
138 }
139
140 // Проверяем дополнительные изображения
141 if (currentMotorcycle.images && currentMotorcycle.images.length > 0) {
142     const newImages = images || [];
143     currentMotorcycle.images.forEach(img => {
144         if (img.startsWith('http://localhost:5000/uploads/') && !newImages.includes(img)) {
145             imagesToDelete.push(img);
146         }
147     });
148 }
149
150 // Удаляем неиспользуемые изображения
151 for (const imageUrl of imagesToDelete) {
152     try {
153         const filename = imageUrl.split('/').pop();
154         if (filename) {
155             const filePath = path.join(__dirname, '../uploads', filename);
156             if (fs.existsSync(filePath)) {
157                 fs.unlinkSync(filePath);
158             }
159         }
160     } catch (error) {
161         console.error('Error deleting old image file:', error);
162     }
163 }
164
165
166 // Если изменилось название, генерируем новый slug
167 let finalUpdateData: any = {
168     name: name ?? currentMotorcycle?.name,
169     price: price !== undefined ? parseFloat(price) : currentMotorcycle?.price,
170     shortDescription: shortDescription ?? currentMotorcycle?.shortDescription,
171     detailedDescription: detailedDescription ?? currentMotorcycle?.detailedDescription,
172     mainImage: mainImage ?? currentMotorcycle?.mainImage,
173     images,
174     stock: stock !== undefined ? parseInt(stock) || 0 : currentMotorcycle?.stock,
```

Рис. 5. Обновление данных товара и удаление устаревших изображений

Удаление товаров также осуществляется через защищённые административные маршруты и сопровождается каскадной очисткой связанных ресурсов, включая изображения. Это позволяет гарантировать целостность данных и избежать появления «висячих» файлов или некорректных ссылок в пользовательском интерфейсе. Пример реализации операции удаления товара приведён на рисунке (рис. 6).

```
550 // DELETE /admin/details/:id - удалить запчасть
551 router.delete('/details/:id', async (req, res): Promise<void> => {
552   try {
553     const { id } = req.params;
554
555     // Получаем запчасть перед удалением, чтобы удалить связанные изображения
556     const detail = await prisma.detail.findUnique({
557       where: { id: parseInt(id) }
558     });
559
560     if (detail) {
561       // Удаляем изображения запчасти
562       const imagesToDelete = [];
563       if (detail.mainImage && detail.mainImage.startsWith('http://localhost:5000/uploads/')) {
564         imagesToDelete.push(detail.mainImage);
565       }
566       if (detail.images && detail.images.length > 0) {
567         detail.images.forEach(img => {
568           if (img.startsWith('http://localhost:5000/uploads/')) {
569             imagesToDelete.push(img);
570           }
571         });
572       }
573
574       // Удаляем файлы изображений
575       for (const imageUrl of imagesToDelete) {
576         try {
577           const filename = imageUrl.split('/').pop();
578           if (filename) {
579             const filePath = path.join(__dirname, '../../uploads', filename);
580             if (fs.existsSync(filePath)) {
581               fs.unlinkSync(filePath);
582             }
583           }
584         } catch (error) {
585           console.error('Error deleting image file:', error);
586         }
587       }
588     }
589   }
590 }
```

Рис. 6. Реализация операции удаления товара в административной части

Серверная часть системы тесно интегрирована с административной панелью, реализованной на React. Администратор взаимодействует с товарами через таблицы, формы и модальные окна, отправляя запросы к административным API с использованием JWT-токена. Такой подход обеспечивает удобство управления каталогом и позволяет в реальном времени отражать изменения в базе данных. Итоговый вид административной панели управления товарами представлен на рисунке (рис. 7, 8).

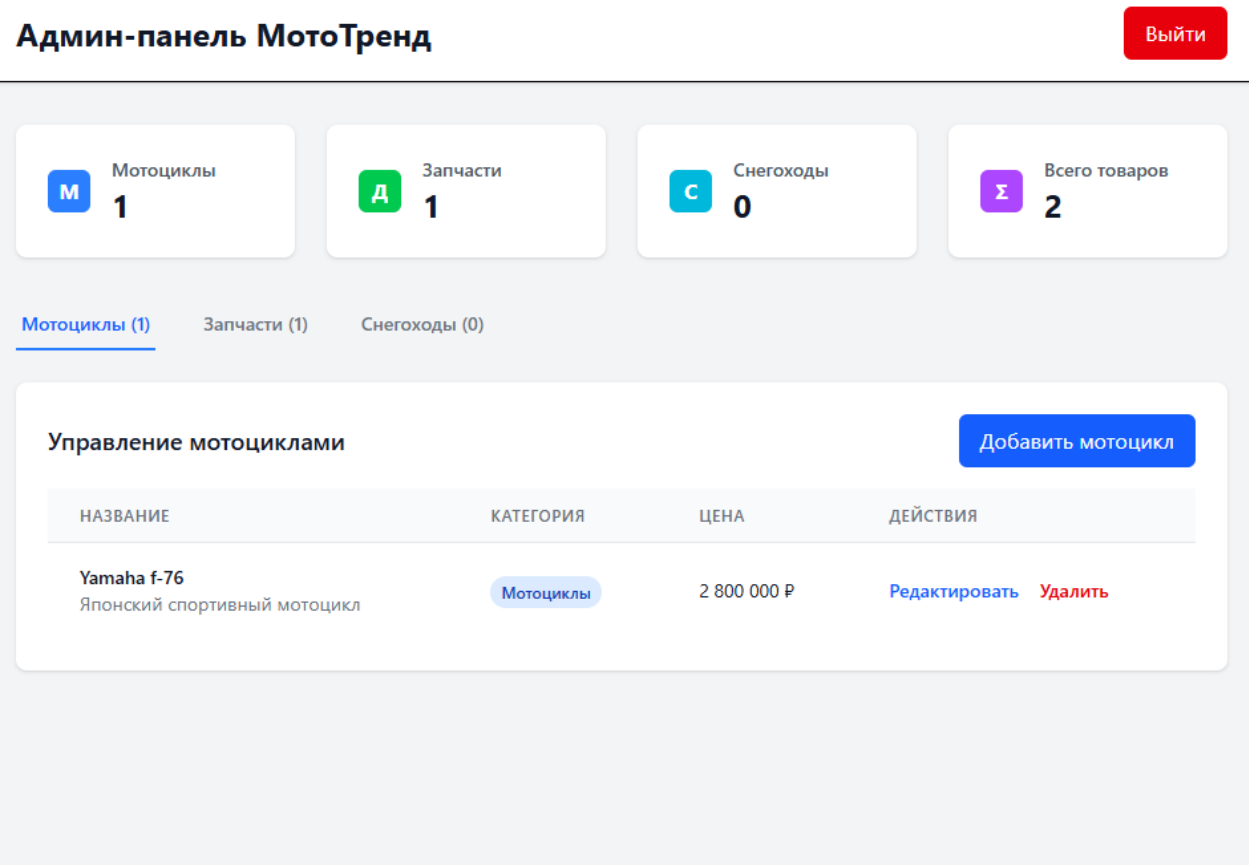


Рис. 7. Интерфейс административной панели управления товарами

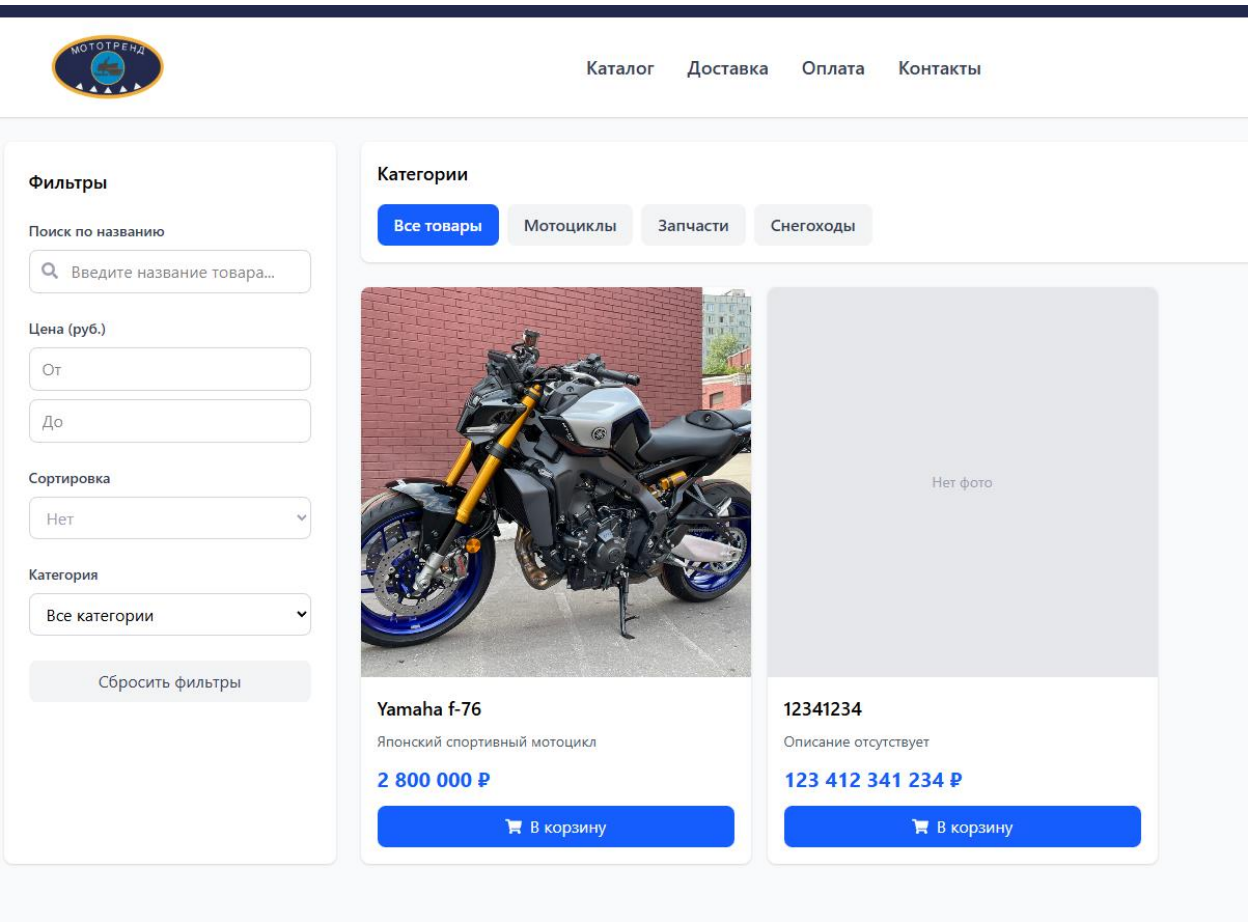


Рис. 8. Вывод товара в каталоге

4 Выводы

В результате работы была разработана система управления товарами (CRUD) для интернет-магазина, обеспечивающая эффективное администрирование товарного каталога. Реализованное решение позволяет безопасно выполнять операции создания, обновления и удаления товаров с использованием серверных API и ORM Prisma. Применение ролевой авторизации и модульной архитектуры повысило надёжность системы и упростило её дальнейшее сопровождение и расширение.

Библиографический список

1. Ковцур М.М., Филимонов В.Е., Щёголев Е.К. Исследование подходов для реализации многопользовательского React-приложения на языке TypeScript // Мобильный бизнес: перспективы развития и реализации систем радиосвязи в России и за рубежом. Материалы 53-й Международной конференции. Москва, 2024. С. 58–61.
2. Дудак А.А. Сравнительный анализ инструментов разработки для систем управления проектами: преимущества стек-технологий TypeScript и React // Новая наука: от идеи к результату. 2024. № 9. С. 32–40.
3. Хайруллин Д.А., Богданов М.Р. Изучение использования TypeScript в React-приложениях, преимущества статической типизации, подходы к типизации компонентов, хуков и состояния // Международный журнал гуманитарных и естественных наук. 2024. № 12-3 (99). С. 233–238.
4. Удальцов К.Р. Формализация шаблонов безопасных pull request-ов для React-приложений с TypeScript в командах с Git Flow // Актуальные проблемы социально-гуманитарного и научно-технического знания. 2025. № 4 (44). С. 35–36.
5. Воронко Т.М. Создание React-библиотеки с использованием TypeScript // Электронные системы и технологии. Материалы 58-й научной конференции аспирантов, магистрантов и студентов БГУИР. Минск, 2022. С. 200–202.