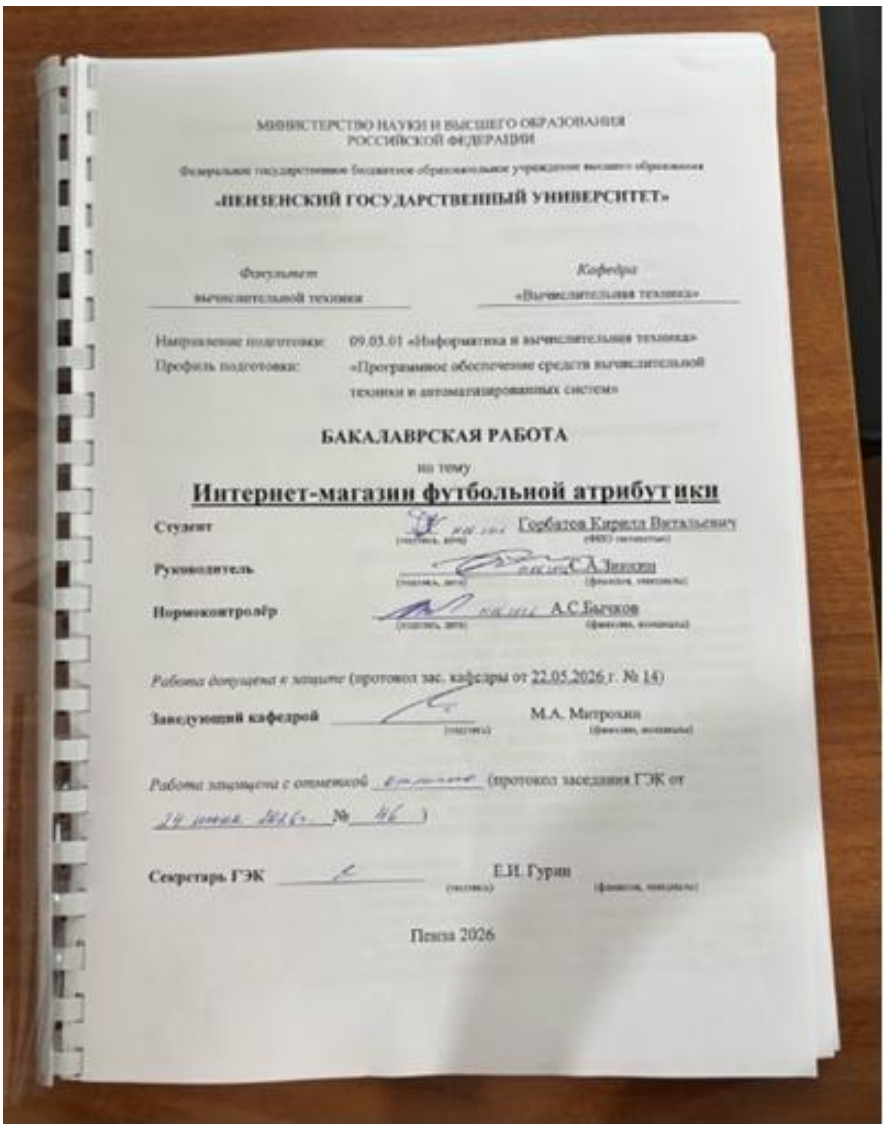


МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

Интернет-магазин футбольной атрибутики

Студент (подпись, дата) Горбатов Кирилл Витальевич (ФИО полностью)

Руководитель (подпись, дата) С.А.Зинкин (фамилия, инициалы)

Нормоконтролёр (подпись, дата) А.С.Бычков (фамилия, инициалы)

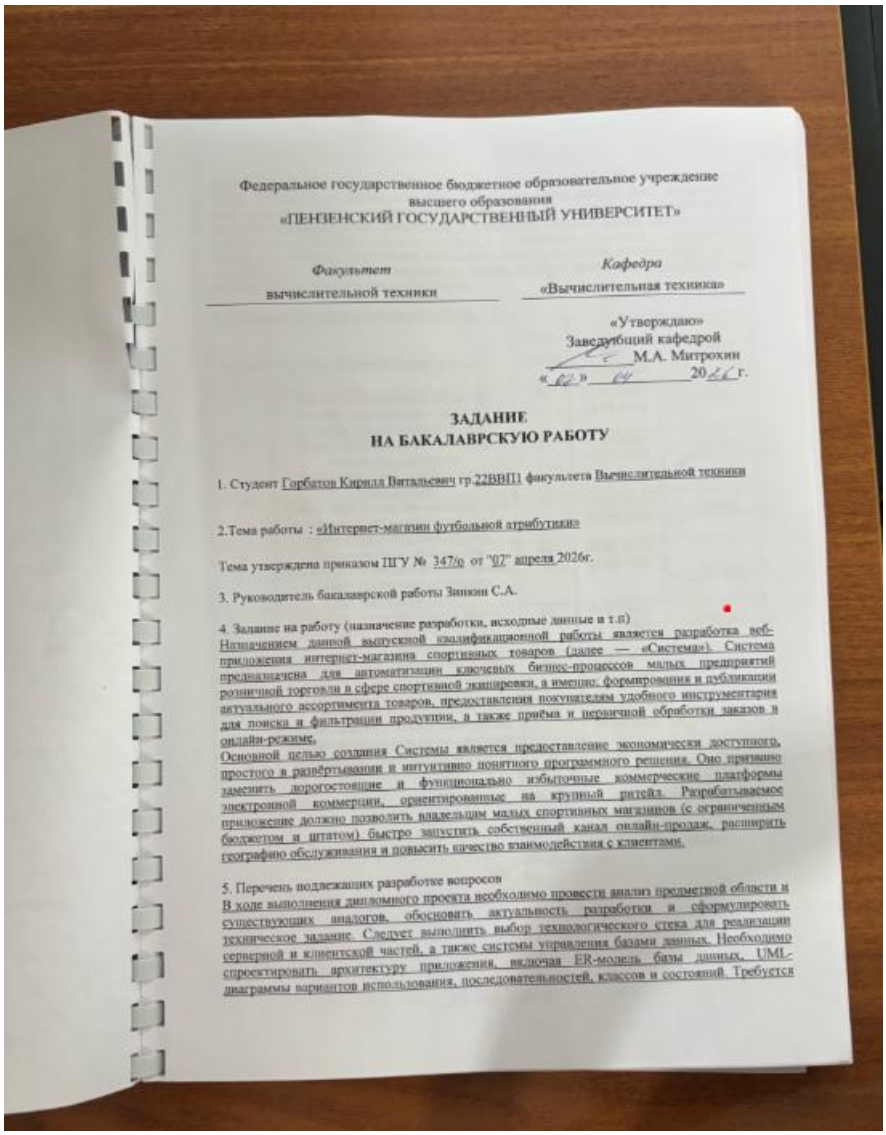
Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой (подпись) М.А. Митрохин (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 19 июня 2026 г. № 46)

Секретарь ГЭК (подпись) Е.И. Гурин (фамилия, инициалы)

Пенза 2026



Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«02» 04 2026 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Горбатов Кирилл Витальевич гр.22ВВП1 факультета Вычислительной техники

2.Тема работы : «Интернет-магазин футбольной атрибутики»

Тема утверждена приказом ПГУ № 347/о от "07" апреля 2026г.

3. Руководитель бакалаврской работы Зинкин С.А.

4. Задание на работу (назначение разработки, исходные данные и т.п)
Назначением данной выпускной квалификационной работы является разработка веб-приложения интернет-магазина спортивных товаров (далее — «Система»). Система предназначена для автоматизации ключевых бизнес-процессов малых предприятий розничной торговли в сфере спортивной экипировки, а именно: формирования и публикации актуального ассортимента товаров, предоставления покупателям удобного инструментария для поиска и фильтрации продукции, а также приёма и первичной обработки заказов в онлайн-режиме.
Основной целью создания Системы является предоставление экономически доступного, простого в развёртывании и интуитивно понятного программного решения. Оно призвано заменить дорогостоящие и функционально избыточные коммерческие платформы электронной коммерции, ориентированные на крупный ритейл. Разрабатываемое приложение должно позволить владельцам малых спортивных магазинов (с ограниченным бюджетом и штатом) быстро запустить собственный канал онлайн-продаж, расширить географию обслуживания и повысить качество взаимодействия с клиентами.

5. Перечень подлежащих разработке вопросов
В ходе выполнения дипломного проекта необходимо провести анализ предметной области и существующих аналогов, обосновать актуальность разработки и сформулировать техническое задание. Следует выполнить выбор технологического стека для реализации серверной и клиентской частей, а также системы управления базами данных. Необходимо спроектировать архитектуру приложения, включая ER-модель базы данных, UML-диаграммы вариантов использования, последовательностей, классов и состояний. Требуется

разработать серверную часть на базе Spring Boot с реализацией REST API и интеграцией с базой данных через Spring Data JPA. Клиентская часть должна быть реализована с использованием React с компонентной архитектурой, адаптивным интерфейсом и поддержкой динамической фильтрации и сортировки товаров. Кроме того, необходимо реализовать функционал корзины покупок с сохранением состояния в localStorage и оформление заказа с сохранением контактных данных. В составе пояснительной записки должны быть представлены разделы по охране труда и экономическому обоснованию проекта, а также руководство пользователя.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области и анализ существующих аналогов (Lamoda, Спортмастер, FreeRide)	40	17.04.2026-22.04.2026	
2	Анализ требований и постановка задачи (формирование функциональных и технических требований)	20	23.04.2026-25.04.2026	
3	Выбор технологий, применяемых при проектировании и разработке (Java / Spring Boot, PostgreSQL, React)	30	26.04.2026-30.04.2026	
4	Проектирование архитектуры системы, базы данных (ER-диаграмма) и пользовательского интерфейса (UML-диаграммы)	60	01.05.2026-05.05.2026	
5	Разработка серверной части (Spring Boot, REST API, Spring Data JPA, Flyway миграции)	80	06.05.2026-11.05.2026	
6	Разработка клиентской части (React, компоненты, Context API, адаптивная верстка, интеграция с API)	80	12.05.2026-17.05.2026	
7	Интеграция компонентов, тестирование и отладка (тест-кейсы, обработка исключительных ситуаций)	40	18.05.2026-20.05.2026	
8	Оформление пояснительной записки	20	21.05.2026	

Дата выдачи задания "07" 04 2026 г.

Руководитель бакалаврской работы _____ С.А. Зинкин
(подпись, дата)

Задание к исполнению принял студент _____ К.В. Горбатов
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Л.Р. Фисенко
(подпись, дата)
21.06.2026

Реферат

Бакалаврская работа содержит 134 листа, 17 рисунков, 1 таблица, 11 источников и 3 приложения.

ИНТЕРНЕТ-МАГАЗИН, СПОРТИВНЫЕ ТОВАРЫ, КАТАЛОГ ТОВАРОВ, ВЕБ-ПРИЛОЖЕНИЕ, БАЗА ДАННЫХ, REACT, SPRING BOOT, POSTGRESQL.

Цель работы — разработка веб-приложения интернет-магазина спортивных товаров для автоматизации процессов представления ассортимента, фильтрации и оформления заказов.

Для достижения поставленной цели было разработано клиент-серверное веб-приложение, отвечающее следующим поставленным задачам: отображение многоуровневого каталога товаров с разбивкой по категориям, фильтрация и сортировка товаров по различным характеристикам, просмотр детальной информации о товаре, работа с корзиной покупок, оформление заказа с сохранением контактных данных покупателя, динамическое управление содержимым страниц и пунктами меню через базу данных, отображение контактной информации магазина.

В результате проведенной работы разработано веб-приложение с клиентской частью на библиотеке React (JavaScript) и серверной частью на фреймворке Spring Boot (Java). Взаимодействие между клиентом и сервером реализовано посредством REST API. Для хранения и управления данными применена СУБД PostgreSQL, для версионирования схемы базы данных использован инструмент миграций Flyway.

Содержание

Введение 5
1 Система автоматизации работы интернет-магазина спортивных товаров 7
1.1 Анализ предметной области 7
1.1.1 Актуальность разработки 7
1.1.2 Анализ существующих аналогов 9
1.1.3 Обоснование актуальности разработки программы 23
1.1.4 Постановка задачи 23
1.2 Выбор технологий и средств разработки приложения 28
1.2.1 Выбор языка программирования 28
1.2.2 Выбор СУБД 29
1.2.3 Графический клиент pgAdmin 4 31
1.2.4 Выбор среды разработки 32
1.2.5 Выбор графической библиотеки 33
1.2.6. Обмен данными JSON 35
1.3 Проектирование информационной системы 35
1.3.1 Паттерн MVC 35
1.3.2 Архитектура приложения 37
1.3.3 Проектирование базы данных 43
1.4 Разработка приложения 56
1.4.1 Реализация БД 56
1.4.2 Описание программы 60
1.4.3 Руководство пользователя 79
1.4.4 Тестирование и отладка 89
2 Охрана труда 94
Заключение 97
Список использованных источников 99
Приложение А – UML-диаграммы 100
Приложение Б – Листинг программы 105
Приложение В – Презентация 136

Перечень сокращений и обозначений

В настоящей работе приняты следующие сокращения и обазначения

АС — автоматизированная система

БД — база данных

Веб-приложение — клиент-серверное приложение, в котором клиентом выступает браузер, а сервером — веб-сервер, взаимодействующий с пользователем через сеть Интернет

Клиент-серверная архитектура — архитектурный шаблон, разделяющий систему на две части: клиентскую, отвечающую за пользовательский интерфейс, и серверную, обрабатывающую бизнес-логику и данные

СУБД — система управления базами данных

API (Application Programming Interface) — программный интерфейс приложения, описывающий способы взаимодействия одной программы с другой

CORS (Cross-Origin Resource Sharing) — механизм, позволяющий веб-страницам запрашивать ресурсы с домена, отличного от домена текущей страницы

CSS (Cascading Style Sheets) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки

FK (Foreign key) — внешний ключ, понятие теории реляционных баз данных, относящееся к ограничениям целостности базы данных

HTML (HyperText Markup Language) — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере

HTTP (HyperText Transfer Protocol) — протокол передачи данных, используемый для передачи веб-страниц и связанных с ними ресурсов

ID (Identifier) — уникальный идентификатор объекта

IDE (Integrated Development Environment) — интегрированная среда разработки

JSON (JavaScript Object Notation) — текстовый формат обмена данными, основанный на синтаксисе языка JavaScript

JSX (JavaScript XML) — расширение синтаксиса языка JavaScript, используемое в библиотеке React для описания структуры пользовательского интерфейса

ORM (Object-Relational Mapping) — технология программирования, связывающая базы данных с концепциями объектно-ориентированных языков программирования

PK (Primary key) — первичный ключ, понятие теории реляционных баз данных, относящееся к ограничениям целостности базы данных

REST API (Representational State Transfer API) — архитектурный стиль взаимодействия компонентов распределённого приложения в сети, основанный на HTTP-протоколе

SPA (Single Page Application) — одностраничное приложение, веб-приложение, работающее в пределах одной HTML-страницы и динамически обновляющее содержимое без перезагрузки

SQL (Structured Query Language) — язык структурированных запросов, применяемый для управления данными в реляционных базах данных

UI (User Interface) — пользовательский интерфейс, совокупность средств, с помощью которых пользователь взаимодействует с программой

URL (Uniform Resource Locator) — единообразный указатель ресурса, адрес веб-страницы или файла в сети Интернет

Введение

Цифровая трансформация затронула практически все направления экономической деятельности, не обошла она стороной и розничную торговлю. С каждым годом всё больше покупателей предпочитают заказывать товары через интернет — это удобно, не нужно никуда идти, а выбор огромен. Рынок спортивной экипировки в этом смысле не отличается от остальных: здесь представлены тысячи наименований, которые различаются по назначению, размерам, материалам, цветам и ещё множеству характеристик.

Современные программы для торговли позволяют автоматизировать рутинные операции — учёт товаров, обработку заказов, хранение информации о покупателях. Системы такого рода берут на себя большую часть административной работы, благодаря чему у руководства остаётся больше времени на планирование и развитие бизнеса.

Для магазина спортивных товаров сайт становится круглосуточной витриной, доступной из любой точки. Кроме показа товаров, такие платформы умеют подбирать продукцию по фильтрам, собирать статистику о том, что покупают чаще всего, и проводить заказы от начала до конца.

Если посмотреть на рынок в целом, то крупные спортивные сети уже давно работают с мощными системами интернет-торговли. Затраты на лицензии, настройку под себя и техническую поддержку в их случае оправданы — они окупаются за счёт больших объёмов продаж и широкого охвата покупателей.

Для небольших региональных магазинов и локальных точек продажи спортивных товаров деньги становятся главным препятствием. Расходы на покупку и поддержку полноценных коммерческих систем часто оказываются просто неподъёмными для малого бизнеса — они несоразмерны с оборотами. К этому добавляется нехватка квалифицированных ИТ-специалистов и времени у самих владельцев, которым приходится заниматься всем сразу. В итоге многие мелкие предприниматели в этой нише либо продолжают торговать только офлайн,

либо довольствуются устаревшими сайтами, которые неудобны для пользователей и криво выглядят на телефонах и планшетах.

Сложившаяся ситуация подталкивает к созданию другого инструмента — веб-приложения, рассчитанного именно на нужды маленьких магазинов спортивных товаров. Нужно разработать такое решение, в котором будут все основные функции современного интернет-магазина: понятный каталог, фильтры, корзина для покупок — но при этом оно останется простым в установке и не потребует мощного оборудования. Этими обстоятельствами и продиктована тема выпускной квалификационной работы — создание веб-приложения интернет-магазина спортивных товаров.

В ходе выполнения работы решались следующие задачи:

– разобраться в специфике предметной области и посмотреть, что уже есть на рынке;

– спроектировать схему взаимодействия клиента и сервера;

– написать серверную часть на Java с использованием Spring Boot и сделать REST API;

– сделать клиентскую часть на React, чтобы страницы обновлялись динамически, без перезагрузки;

– спроектировать реляционную базу данных для хранения информации о товарах, структуре разделов и заказах;

– провести тестирование готового продукта.

Практическая польза работы в том, что владельцы малых спортивных магазинов получат готовое к применению программное средство. Оно поможет расширить круг покупателей, улучшить качество обслуживания за счёт быстрого доступа к информации о товарах и упростить приём заказов. В условиях растущей конкуренции и повсеместного ухода торговли в цифровую среду это особенно важно.

1 Система автоматизации работы интернет-магазина спортивных товаров

1.1 Анализ предметной области

1.1.1 Актуальность разработки

Торговля через интернет давно стала привычным способом продаж для самых разных товаров, и спортивная экипировка здесь не исключение. Покупатель, который хочет купить кроссовки, мяч или тренировочный костюм, рассчитывает увидеть перед собой понятный список того, что есть в наличии. Ему нужно быстро отсеять неподходящее по размеру или цвету и без лишних сложностей сообщить продавцу, что он хочет оформить заказ.

Для владельца даже небольшого магазина важно, чтобы весь ассортимент был показан наглядно, а сведения о каждом товаре — название, цена, картинка, описание — были доступны посетителю в любой момент. Если обновлять ассортимент вручную, например правя статичные HTML-страницы, на это уходит уйма времени, и к тому же легко появляются устаревшие или ошибочные данные. Точно так же, если записывать поступающие заказы где придётся — в блокноте, в мессенджере — часть обращений неизбежно теряется, а вместе с ними теряется и прибыль.

Специализированная программа помогает переложить все эти хлопоты на автоматизированную систему. В случае интернет-магазина такая система берёт на себя хранение данных о товарах в одном месте, сама формирует страницы каталога, обрабатывает фильтры, которые задаёт пользователь, и сохраняет оформленные заказы в базе данных.

Крупные сетевые магазины спортивных товаров обычно работают на сложных и дорогих платформах для онлайн-торговли. Такие системы состоят из десятков связанных между собой модулей, а чтобы их развернуть и поддерживать, нужны квалифицированные разработчики и системные администраторы.

Для небольшого регионального магазина, где хозяин часто сам и управляющий, и товаровед, и продавец в одном лице, покупка и настройка тяжёлой коммерческой системы не всегда имеет смысл. Расходы на лицензию и техническую поддержку могут оказаться сравнимы с месячной выручкой, а избыточные функции, которые малому бизнесу просто не нужны, только мешают разобраться в программе.

В результате многие малые предприятия, торгующие спортивными товарами, либо вообще не представлены в интернете, либо обходятся простеньким сайтом-визиткой. На таком сайте покупатель не может ни отфильтровать товары по нужным характеристикам, ни самостоятельно оформить заказ.

Поэтому есть реальная потребность в веб-приложении, которое, с одной стороны, давало бы базовый, но достаточный набор инструментов для онлайн-торговли — просмотр каталога, фильтры и сортировка, корзина, форма отправки заказа. А с другой стороны, такое приложение должно быть простым в установке и не требовать серьёзных денежных вложений на старте.

Приложение, которое разрабатывается в рамках этой выпускной квалификационной работы, должно эту потребность закрыть. Вот его ключевые особенности:

– все данные о товарах, категориях и пунктах меню хранятся в базе данных, что позволяет управлять содержимым магазина из одного места;

– страницы каталога формируются динамически, можно фильтровать товары сразу по нескольким параметрам — цвет, размер, тип, материал — и сортировать по цене или названию;

– есть корзина, где посетитель собирает выбранные товары и может изменить их список перед тем, как отправить заказ;

– форма оформления заказа с обязательным указанием номера телефона для связи с покупателем;

– серверная часть сделана отдельно от интерфейса и предоставляет REST API — это упрощает возможную доработку системы в будущем.

Перечисленных функций достаточно, чтобы покрыть первоочередные нужды малого спортивного магазина, который хочет начать принимать заказы через интернет. При этом в системе нет ничего лишнего — только то, что действительно нужно, без накруток, усложняющих работу и повышающих стоимость эксплуатации.

1.1.2 Анализ существующих аналогов

Среди программных решений и онлайн-платформ, автоматизирующих процесс продажи спортивных товаров через интернет, можно выделить следующие крупные проекты:

– интернет-витрина и мобильное приложение Lamoda (раздел «Спорт»);

– интеграционное решение для маркетплейсов на примере кейса Desport (ранее Decathlon) на Wildberries;

– интернет-магазин федеральной розничной сети «Спортмастер».

Рассмотрим функциональные возможности и особенности каждого из перечисленных аналогов.

Интернет-платформа Lamoda (раздел «Спорт»).

Lamoda представляет собой крупную мультибрендовую платформу, объединяющую торговлю одеждой, обувью, аксессуарами, а также товарами для спорта и активного отдыха. Спортивный раздел насчитывает несколько тысяч наименований продукции, сгруппированных по категориям (мужская/женская спортивная одежда, обувь, инвентарь).

Ключевые функциональные элементы платформы, имеющие отношение к рассматриваемой предметной области :

– многоуровневая система фильтрации. Посетителю доступен отбор товаров по широкому спектру параметров: пол, бренд, размерный ряд, цветовая гамма, диапазон цен, страна-производитель. Имеется возможность отображения только товаров со скидкой;

– сортировка выдачи. Результаты поиска могут быть упорядочены по популярности, новизне, возрастанию или убыванию цены;

– карточка товара. Содержит детальное описание, фотографии в высоком разрешении, таблицу размеров, информацию о составе материала и рекомендации по уходу, а также отзывы покупателей;

– корзина и оформление заказа. Реализован стандартный механизм накопления выбранных позиций с последующим переходом к форме оформления, где указываются контактные данные, адрес и способ доставки.

К преимуществам Lamoda можно отнести широкий ассортимент, отлаженную логистику и интуитивно понятный интерфейс. Однако для малого бизнеса использование данной платформы в качестве собственного канала продаж не представляется возможным, поскольку Lamoda является самостоятельным ритейлером, а не конструктором магазинов. Кроме того, выход на подобные маркетплейсы сопряжён с высокими комиссиями и жёсткими требованиями к поставщикам.

На рисунке 1 представлен внешний вид сайта.

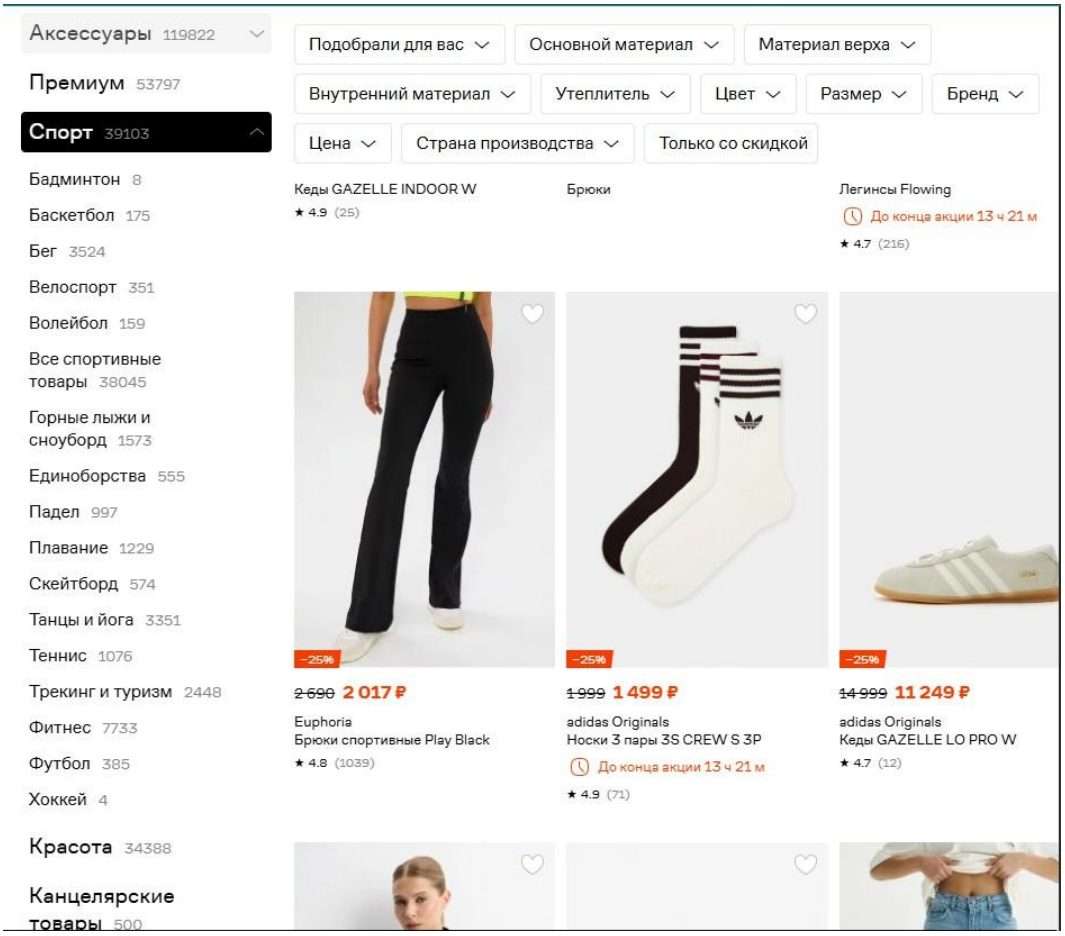


Рисунок 1 – Главный вид сайта «Lamoda»

Интернет-магазин сети «FreeRide».

«FreeRide» представляет собой интернет-магазин сети спортивных магазинов, специализирующейся на продаже велосипедов, сноубордов, горнолыжного снаряжения, а также сопутствующей одежды и аксессуаров. Проект был реализован с целью расширения географии продаж и улучшения клиентского опыта .

Рассмотрим функциональные возможности данной системы:

– интеграция с учётными системами. Магазин интегрирован с CRM и ERP-системами на базе платформы «1С-Битрикс», что позволяет автоматизировать обмен данными о товарах, заказах и складских остатках. Сотрудникам не требуется вручную переносить информацию из одной программы в другую;

– гибкая структура каталога. Товары распределены по категориям и подкатегориям с учётом специфики ассортимента. Например, посетитель может

последовательно сужать область поиска: «Велосипеды» → «Горные» → «Двухподвесы», либо воспользоваться фильтрами по бренду, размеру рамы, типу тормозов и другим техническим характеристикам;

– настройка способов оплаты и доставки. Реализована интеграция с платёжными системами и службами доставки. Покупатель может выбрать наиболее удобный для себя вариант получения заказа (курьерская доставка, пункт выдачи, самовывоз из розничного магазина сети);

– оптимизация контента. На этапе проектирования был проведён анализ поискового спроса, на основе которого оптимизирована структура сайта и наполнение страниц. Это позволило улучшить видимость магазина в поисковых системах и привлечь целевую аудиторию.

К достоинствам данного решения можно отнести высокую степень автоматизации бизнес-процессов за счёт интеграции с товароучётной системой, широкие возможности для масштабирования и профессиональный подход к юзабилити.

Однако, наряду с преимуществами, система обладает и рядом недостатков, существенных с точки зрения малого бизнеса:

– высокая стоимость разработки и внедрения. Бюджет аналогичного проекта, выполненного студией-разработчиком, составил порядка 750 000 рублей, что является неподъёмной суммой для начинающего предпринимателя или небольшого регионального магазина;

– зависимость от платной платформы. Функционирование магазина обеспечивается системой «1С-Битрикс», требующей приобретения лицензии и регулярной оплаты за её продление;

– сложность администрирования. Для настройки и поддержания работоспособности всех интегрированных модулей (CRM, складской учёт, обмен с 1С) требуется привлечение квалифицированного специалиста, что влечёт за собой дополнительные ежемесячные расходы.

Таким образом, несмотря на свою функциональную полноту, интернет-магазин, построенный по модели «FreeRide», оказывается избыточным и

экономически нецелесообразным решением для небольшой торговой точки, делающей первые шаги в онлайн-торговле.

На рисунке 2 представлен внешний вид сайта «FreeRide».

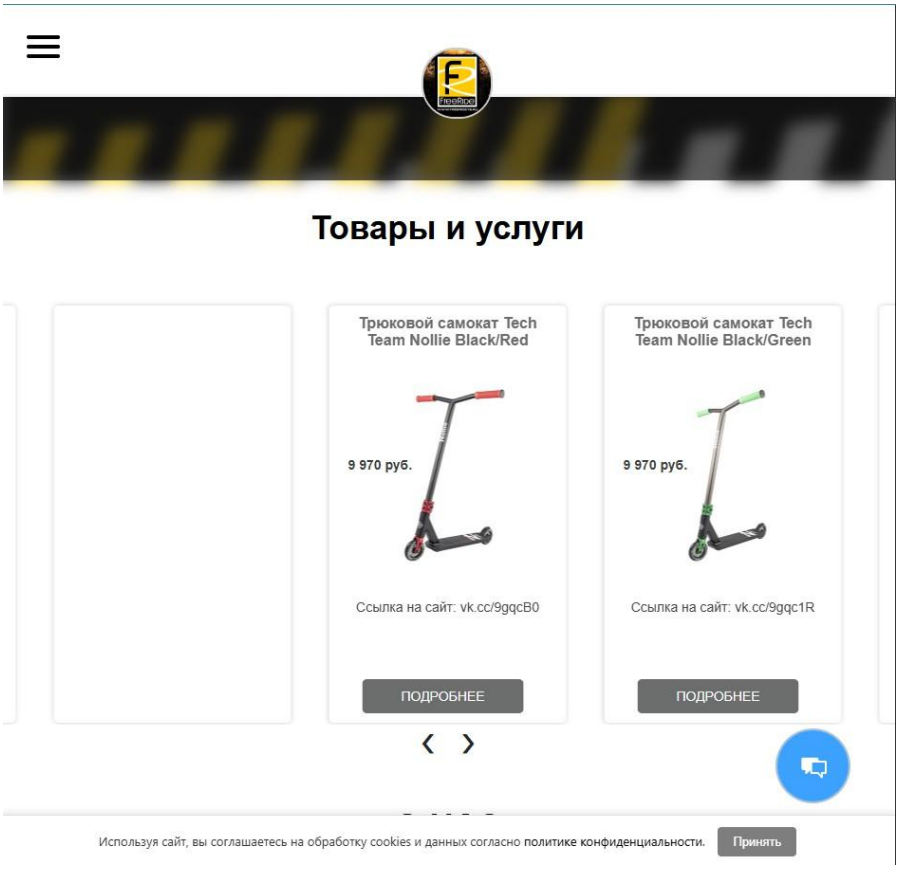


Рисунок 2 – «FreeRide»

Интернет-магазин сети «Спортмастер».

«Спортмастер» является крупнейшим в России ритейлером спортивных товаров, чья доля на рынке, по данным аналитических исследований, достигает 79% . Помимо широкой сети физических супермаркетов, компания располагает развитым интернет-магазином.

Основные характеристики веб-витрины «Спортмастер»:

– иерархический каталог. Товары разделены на крупные категории (обувь, одежда, инвентарь, тренажёры), внутри которых реализована детализация по видам спорта и брендам;

– расширенные фильтры. Помимо базовых параметров (размер, цвет, цена), доступна фильтрация по специфическим характеристикам, зависящим от категории (например, для беговых кроссовок — тип пронации стопы, сезонность, вес модели);

– программа лояльности. Реализован личный кабинет покупателя с возможностью накопления и списания бонусных баллов, отслеживания истории заказов и статуса текущих доставок;
– омниканальность. Система интегрирована с офлайн-магазинами сети: покупатель может проверить наличие товара в конкретной точке продаж и зарезервировать его для примерки.

С точки зрения функциональной полноты, интернет-магазин «Спортмастер» является эталонным решением для крупного ритейла. Вместе с тем, его программная реализация представляет собой сложный комплекс, включающий собственную CRM-систему, складскую логистику, модули аналитики и колл-центр. Воспроизведение подобной архитектуры в рамках дипломного проекта нецелесообразно и избыточно для задач малого предприятия.

Сравнительный анализ и обоснование собственной разработки.

Все рассмотренные аналоги, несомненно, являются успешными коммерческими продуктами, однако их применение для оснащения небольшого регионального магазина спортивных товаров затруднено по следующим причинам:

– избыточность функционала. Lamoda, Wildberries и «Спортмастер» ориентированы на обработку тысяч заказов в сутки и управление складскими запасами федерального масштаба. Для малого бизнеса значительная часть этих возможностей останется невостребованной;

– финансовый барьер. Стоимость лицензирования, внедрения и технической поддержки промышленных платформ электронной коммерции или индивидуальной разработки интеграционного слоя к маркетплейсам непосильна для предприятия с ограниченным оборотом;

– сложность администрирования. Эксплуатация многофункциональных систем требует наличия в штате квалифицированного ИТ-персонала либо заключения дорогостоящих договоров аутсорсинга.

Таким образом, сохраняется потребность в создании компактного веб-приложения, которое, с одной стороны, предоставит базовые механизмы онлайн-торговли (навигация по каталогу, фильтрация по ключевым характеристикам, корзина, форма заказа), а с другой — будет лишено перечисленных недостатков. Разрабатываемая в рамках данной ВКР система призвана заполнить эту нишу.

На рисунке 3 представлен внешний вид сайта «Спортмастер».

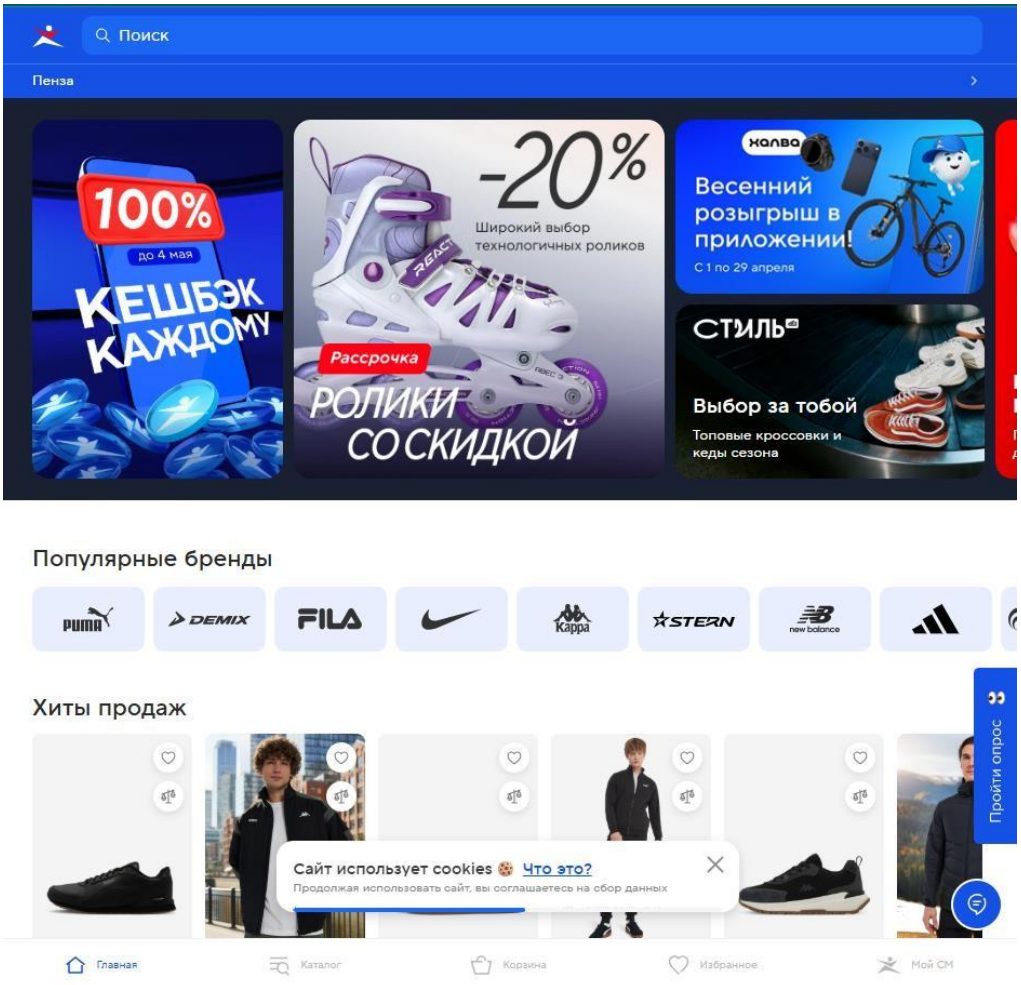


Рисунок 3 – Внешний вид «Спортмастер»

В таблице 1 представлен анализ аналогов.

Таблица 1 – Анализ аналогов

Название системы	Основные возможности	Преимущества	Недостатки
Lamoda (раздел «Спорт»)	Многоуровневый каталог, расширенная фильтрация по бренду, размеру, цвету, цене; сортировка выдачи; карточка товара с детальным описанием и отзывами;	Широкий ассортимент, отлаженная логистика, интуитивно понятный интерфейс, высокая узнаваемость бренда.	Закрытая экосистема, высокие комиссии и жёсткие требования к поставщикам при выходе на маркетплейс.
FreeRide	Гибкая структура каталога; интеграция с CRM и ERP-системами; настройка способов оплаты и доставки.	Высокая степень автоматизации бизнес-процессов за счёт интеграции с 1С; профессиональный подход к юзабилити; возможности для масштабирования.	Высокая стоимость разработки зависимость от платной платформы 1С-Битрикс; необходимость квалифицированного администрирования.

Название системы	Основные возможности	Преимущества	Недостатки
Спортмастер	Иерархический каталог с разбивкой по видам спорта; расширенные фильтры по специфическим характеристикам; личный кабинет с бонусной программой и историей заказов;	Функциональная полнота, эталонное решение для крупного ритейла, интеграция с программой лояльности и складскими остатками розничной сети.	Сложная программная реализация (CRM, складская логистика, колл-центр); избыточность и высокая стоимость владения для малого бизнеса.
«Sportwear Store»	Динамический каталог товаров по категориям; фильтрация по цвету, размеру, типу, материалу; сортировка по цене и наименованию; постраничная навигация; корзина покупок;	Низкая стоимость внедрения и владения; отсутствие лицензионных отчислений; простота развёртывания и нетребовательность к хостингу; наличие базового функционала, достаточного для старта онлайн-продаж малым предприятием.	Требует разработки и настройки; на начальном этапе отсутствуют интеграции с платёжными системами и службами доставки, личный кабинет покупателя и расширенная аналитика.

1.1.3 Обоснование актуальности разработки программы

Анализ, проведённый в предыдущем разделе, показал, что рынок систем для онлайн-торговли устроен так: либо это закрытые маркетплейсы, куда мелкому продавцу сложно попасть, либо дорогие комплексные платформы, рассчитанные на средний и крупный бизнес. Для небольшого регионального магазина спортивных товаров, который только пробует выйти в онлайн, и тот и другой вариант создаёт массу проблем.

Первая проблема — деньги. Промышленные CMS-системы стоят дорого: лицензия, услуги по внедрению, дальнейшая техподдержка — всё это может обойтись в сотни тысяч рублей. Для малого предприятия это сопоставимо с месячной, а то и квартальной выручкой. Ко всему прочему, многие платформы требуют регулярных абонентских отчислений или удерживают комиссию с каждого заказа, из-за чего финансовая нагрузка становится постоянной.

Вторая проблема — лишние функции. Большие интернет-магазины оснащены десятками модулей: складской учёт в масштабах целой сети, стыковка с кучей служб доставки, программы лояльности, умные рекомендации на основе машинного обучения. Но владельцу маленькой торговой точки, где ассортимент состоит из нескольких сотен позиций, а заказы обрабатываются вручную, почти ничего из этого не нужно. Весь этот набор только мешает — путает, отвлекает от действительно важных задач и растягивает время на освоение программы.

Третья проблема — сложность обслуживания. Чтобы поддерживать многофункциональную платформу, нужен квалифицированный ИТ-специалист в штате или договор со сторонней организацией. А в малом бизнесе хозяин обычно сам себе и управляющий, и товаровед, и продавец. Лишние расходы на техническое сопровождение ему просто не по карману.

В итоге на рынке остаётся пустая ниша: нет простого и доступного веб-приложения, которое давало бы базовые, но достаточные для начала онлайн-продаж возможности и при этом было рассчитано именно на нужды малых спортивных магазинов.

Система, разрабатываемая в рамках данной выпускной работы, должна эту нишу заполнить. Основные черты, которые делают проект актуальным и практически полезным:

– небольшие расходы на владение. Продукт создаётся на свободно распространяемых технологиях и фреймворках — Java, Spring Boot, React, PostgreSQL. Покупать дорогие лицензии не нужно. Чтобы запустить систему, хватит обычного виртуального хостинга или недорогого VPS-сервера — его аренда обходится на порядок дешевле, чем абонентская плата за коммерческие платформы;

– лёгкость установки и настройки. Никаких сложных процедур. Серверная часть запускается одним JAR-файлом. Клиентская часть — это просто набор статических файлов, готовых к размещению на любом веб-сервере. Наполнение базы данных начальными таблицами и данными делается автоматически с помощью Flyway;

– ничего лишнего. В системе только то, что действительно нужно маленькому магазину на старте: каталог товаров с понятной структурой, удобные фильтры и сортировка, корзина для покупателя и форма заказа с сохранением контактов. Из-за того что нет нагромождения второстепенных функций, интерфейс остаётся простым и понятным, а времени на обучение персонала уходит минимум;

– гибкое управление контентом. Все данные — товары, категории, пункты меню, контактная информация — хранятся в одном месте, в базе данных. Чтобы изменить ассортимент или поправить служебную информацию, не нужно лезть в программный код. Всё можно сделать через обычный графический клиент вроде pgAdmin, с которым справится даже человек с минимальной технической подготовкой;

– работает везде и выглядит нормально на любом экране. Серверная часть на Java запускается под любой операционной системой, где есть виртуальная машина Java — Windows, Linux, macOS. Клиентское приложение на React построено как одностраничное приложение и подстраивается под экраны разного размера;

– можно развивать дальше. Архитектура разделяет клиентскую и серверную части, а общаются они через чётко описанный REST API. Это хорошая основа для будущих доработок. Со временем в систему можно добавить личный кабинет, интеграцию с платёжными системами, панель администратора с графическим интерфейсом и другие нужные функции — и для этого не придётся переписывать всё с нуля.

Все перечисленные причины показывают, что разработка собственного веб-приложения — это разумное и обоснованное решение. В отличие от универсальных коммерческих платформ, оно максимально подстроено под конкретные задачи и скромные ресурсы малого предприятия, которое занимается розничной продажей спортивных товаров.

1.1.4 Постановка задачи

Отталкиваясь от анализа предметной области, разбора существующих аналогов и потребностей малых спортивных магазинов, были сформулированы требования к разрабатываемому веб-приложению — и по функционалу, и по технической части.

Что система должна уметь делать:

– показывать каталог товаров. Посетитель должен видеть товары, разбитые по категориям: обувь, мячи, одежда, аксессуары, инвентарь и так далее. Для каждой категории — своя страница с постраничной навигацией, чтобы не вываливать всё скопом;

– фильтровать и сортировать. Пользователю нужно дать возможность отбирать товары по параметрам, которые зависят от категории: цвет, размер, тип изделия, материал. Результаты можно отсортировать по цене — от дешёвых к дорогим или наоборот — и по названию в алфавитном порядке;

– показывать подробности о товаре. Если посетитель кликает по товару из каталога, он попадает на отдельную страницу с полной информацией: название, картинка, цена, подробные характеристики и описание;

– работать с корзиной. Посетитель может складывать понравившиеся товары в корзину, смотреть, что там лежит, и убирать отдельные позиции. Корзина должна сохраняться в локальном хранилище браузера, чтобы после закрытия и повторного открытия страницы всё оставалось на месте;

– оформлять заказ. Должна быть форма, через которую покупатель завершает покупку. Обязательное поле — номер телефона для связи. После отправки система сохраняет заказ в базе: фиксирует дату и время, общую сумму и список товаров;

– управлять меню и контентом через базу данных. Структура бокового меню и содержимое страниц каталога должны подгружаться из базы. Это позволяет менять их без правки кода — ни на клиенте, ни на сервере;

– показывать контакты магазина. Внизу каждой страницы, в футере, должны быть телефон, электронная почта, физический адрес и ссылки на соцсети — VK. Вся эта информация тоже берётся из базы данных;

Технические требования:

– клиент-серверная архитектура. Приложение делится на две части: клиентскую, которая отвечает за интерфейс, и серверную, в которой зашита бизнес-логика и работа с базой данных. Общаются они между собой через REST API, обмениваясь данными в формате JSON;

– простой и понятный интерфейс. Клиентская часть должна быть удобной: нормальная навигация по каталогу, товары в виде карточек с картинкой, названием и ценой, быстрый доступ к фильтрам и корзине;

– адаптивная вёрстка. Расположение элементов должно подстраиваться под размер окна браузера, чтобы сайт нормально выглядел на любых устройствах — и на большом мониторе, и на экране телефона;

– кроссплатформенность серверной части. Сервер пишется на Java с использованием Spring Boot и пакуется в один исполняемый JAR-файл. Это даёт

возможность запускать его на любой операционной системе, где есть Java — Windows, Linux, macOS;

– единое хранилище данных. Все данные — товары, категории, заказы и служебная информация — хранятся в реляционной базе данных под управлением PostgreSQL. Структура базы создаётся и обновляется автоматически с помощью миграций Flyway.

Минимальные требования к аппаратному обеспечению сервера. Для развёртывания и функционирования системы достаточно стандартной конфигурации виртуального выделенного сервера (VPS):

– операционная система: Windows Server 2012 и выше, либо Linux (Ubuntu 20.04 / Debian 11 и выше);

– оперативная память: не менее 1 ГБ ОЗУ (рекомендуется 2 ГБ);

– свободное дисковое пространство: не менее 2 ГБ (с учётом хранения файлов изображений);

– процессор: 1 виртуальное ядро.

Требования к клиентскому рабочему месту. Для доступа к веб-приложению пользователю необходим любой современный браузер (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari актуальных версий) и подключение к сети Интернет. Специальных требований к аппаратной части клиентского компьютера не предъявляется.

На рисунке 4 изображена диаграмма вариантов использования приложения для посетителя интернет-магазина, на рисунке 5 — диаграмма вариантов использования для администратора, управляющего контентом через базу данных.

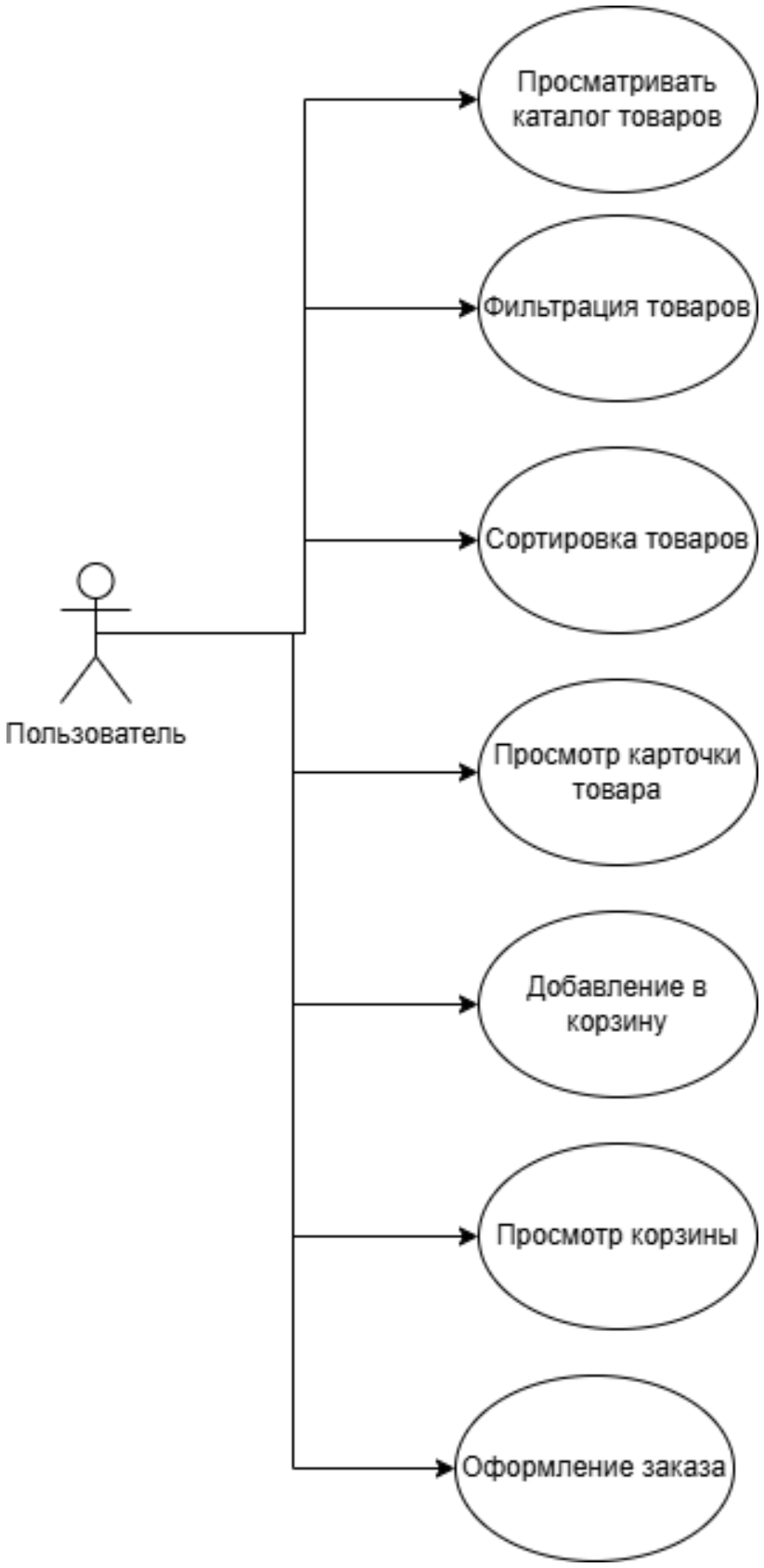


Рисунок 4 – Диаграмма вариантов использования приложения для посетителя интернет-магазина

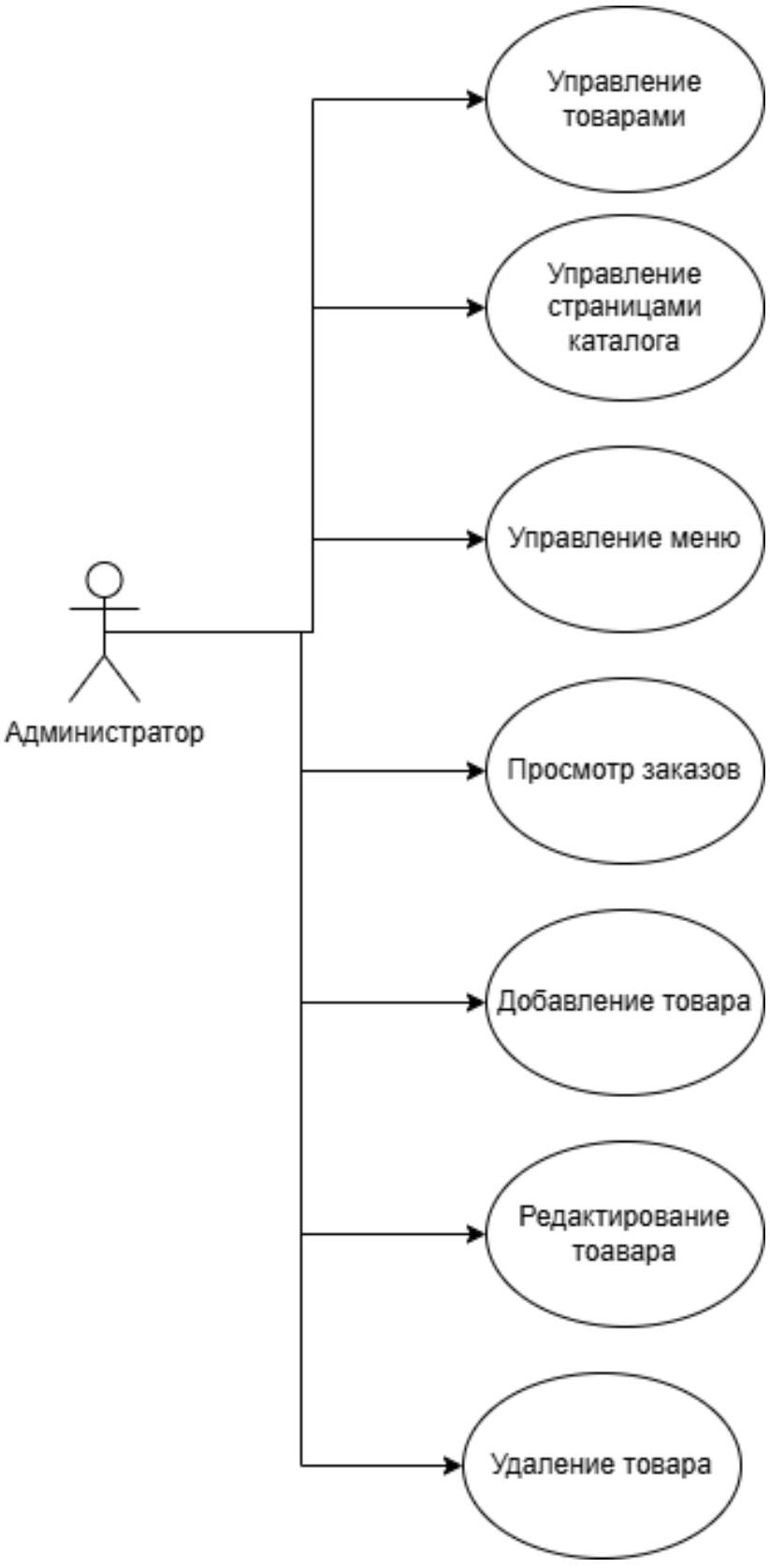


Рисунок 5 – Диаграмма вариантов использования для администратора, управляющего контентом через базу данных

1.2 Выбор технологий и средств разработки приложения

В данном разделе будут описаны технологии и средства разработки, использованные при создании веб-приложения интернет-магазина спортивных товаров.

Ниже приведён перечень ключевых технологий, выбор которых был обоснован в ходе проектирования:

– языки программирования: Java (серверная часть) и JavaScript (клиентская часть);

– фреймворк для серверной части: Spring Boot;

– библиотека для клиентской части: React;

– система управления базами данных: PostgreSQL;

– инструмент миграций базы данных: Flyway;

– среда разработки серверной части: IntelliJ IDEA / Apache NetBeans;

– среда разработки клиентской части: Visual Studio Code;

– инструмент сборки клиентской части: Vite;

– формат обмена данными: JSON.

1.2.1 Выбор языка программирования

Разработка веб-приложения предполагает создание двух независимых компонентов — серверной части (backend) и клиентской части (frontend). Для каждой из них выбор языка программирования осуществлялся отдельно, исходя из специфики решаемых задач и доступного инструментария.

Выбор языка для серверной части (backend)

Для реализации серверной логики рассматривались следующие языки программирования:

Язык программирования Python.

Python — высокоуровневый интерпретируемый язык программирования с динамической типизацией. Он известен своей простотой, лаконичным синтаксисом, богатой стандартной библиотекой и поддержкой множества парадигм программирования (объектно-ориентированное, функциональное).

Python широко применяется в области анализа данных, машинного обучения и веб-разработки. Существуют популярные фреймворки для создания веб-приложений, такие как Django и Flask, которые позволяют быстро разрабатывать серверные API и работать с базами данных.

Однако Python обладает рядом недостатков, которые необходимо учитывать при разработке промышленных приложений. К ним относятся относительно невысокая производительность по сравнению с компилируемыми языками, динамическая типизация, которая может приводить к ошибкам на этапе выполнения, а также определённые сложности с многопоточностью, связанные с механизмом GIL (Global Interpreter Lock). Для задач интернет-магазина, где важна надёжность и типобезопасность, данные ограничения могут оказаться критичными.

Язык программирования PHP.

PHP — широко распространённый серверный язык сценариев, специально разработанный для веб-разработки. На нём написано огромное количество систем управления контентом (WordPress, Joomla, Drupal) и фреймворков (Laravel, Symfony). PHP прост в освоении, имеет низкий порог входа и поддерживается практически всеми хостинг-провайдерами.

Тем не менее, PHP исторически обладает неоднородным синтаксисом и непоследовательной стандартной библиотекой, что усложняет поддержку и развитие крупных проектов. Кроме того, для обеспечения должного уровня безопасности и производительности приложение на PHP требует более тщательной настройки серверного окружения по сравнению с решениями на Java.

Язык программирования Java.

Java — это строго типизированный, объектно-ориентированный язык программирования, изначально разработанный для создания кроссплатформенных приложений. Java-код компилируется в байт-код, который затем выполняется на виртуальной машине Java (JVM). Такой подход обеспечивает переносимость приложений между различными операционными системами без изменения исходного кода.

Для веб-разработки на Java существует экосистема фреймворков, наиболее заметным из которых является Spring Boot. Он предоставляет готовую инфраструктуру для создания REST API, работы с базами данных, обеспечения безопасности и решения многих других типовых задач. Java отличается высокой производительностью, надёжностью, развитой системой типов, позволяющей выявлять ошибки на этапе компиляции, и автоматическим управлением памятью (сборщик мусора).

Учитывая необходимость создания надёжного, масштабируемого и типобезопасного серверного приложения, выбор был сделан в пользу языка Java в связке с фреймворком Spring Boot. Данная связка является индустриальным стандартом для разработки корпоративных веб-приложений и обеспечивает отличную основу для реализации всех поставленных функциональных требований.

Выбор языка и библиотеки для клиентской части (frontend)

Для создания пользовательского интерфейса, работающего в браузере, основным языком является JavaScript (в современной разработке часто используется его надстройка — TypeScript, добавляющая статическую типизацию). В данном проекте использовался чистый JavaScript с расширением синтаксиса JSX.

В качестве основной библиотеки для построения интерфейса была выбрана React.

React — это декларативная JavaScript-библиотека для создания динамических пользовательских интерфейсов. Её ключевыми особенностями являются:

– компонентный подход. Интерфейс строится из изолированных, переиспользуемых компонентов, каждый из которых инкапсулирует свою логику и внешний вид. Это упрощает разработку, отладку и сопровождение больших приложений;

– виртуальный DOM. React работает с легковесным представлением реальной DOM-модели в памяти. При изменении состояния компонента React

эффективно вычисляет минимальный набор изменений и применяет их к реальному DOM. Такой подход обеспечивает высокую производительность и отзывчивость интерфейса;

– однонаправленный поток данных. Данные в приложении передаются от родительских компонентов к дочерним, что делает поведение приложения более предсказуемым и упрощает отслеживание изменений;

– богатая экосистема. Вокруг React сформировалось огромное сообщество разработчиков и множество готовых библиотек, решающих широкий круг задач (маршрутизация, управление состоянием, работа с формами и др.).

В качестве альтернативы рассматривались такие библиотеки и фреймворки, как Vue.js и Angular. Vue.js отличается более низким порогом входа, однако уступает React по размеру экосистемы и количеству доступных вакансий на рынке труда. Angular представляет собой полноценный фреймворк с жёстко заданной структурой, что делает его избыточным для проекта небольшого или среднего масштаба.

Таким образом, с учётом популярности, производительности и гибкости, для реализации клиентской части веб-приложения была выбрана библиотека React.

1.2.2 Выбор СУБД

Для функционирования интернет-магазина критически важным компонентом является система управления базами данных (СУБД), отвечающая за надёжное хранение, структурирование и быстрый поиск информации о товарах, категориях, заказах и служебных сущностях. При выборе СУБД рассматривались три популярных решения, каждое из которых имеет свои сильные и слабые стороны.

СУБД PostgreSQL.

PostgreSQL — это свободно распространяемая объектно-реляционная система управления базами данных с открытым исходным кодом, предоставляющая широкий спектр возможностей для хранения и обработки структурированной информации. Данная СУБД развивается на протяжении более

двух десятилетий и заслужила репутацию одного из самых надёжных и функционально полных решений, доступных на рынке.

Среди ключевых преимуществ PostgreSQL можно выделить:

– полная поддержка стандарта SQL и расширяемость. PostgreSQL не только строго следует стандарту ANSI SQL, но и предоставляет уникальные возможности по созданию собственных типов данных, операторов, агрегатных и оконных функций. Это позволяет гибко адаптировать базу данных под специфические нужды предметной области;

– богатый набор встроенных типов данных. Помимо стандартных числовых и строковых типов, PostgreSQL поддерживает JSON, JSONB, массивы, геометрические типы, диапазоны и полнотекстовый поиск. Для интернет-магазина спортивных товаров, где характеристики продукции могут сильно варьироваться, поддержка гибких структур данных (например, хранение параметров фильтрации в JSONB) является существенным преимуществом;

– надёжность и соответствие принципам ACID. СУБД обеспечивает транзакционную целостность данных, гарантируя, что в случае любого сбоя информация не будет повреждена или потеряна. Это критически важно для учёта заказов и сохранности коммерческой информации;

– высокая производительность при конкурентном доступе. Благодаря реализации многоверсионности (MVCC), PostgreSQL позволяет множеству клиентов одновременно читать и изменять данные, не блокируя друг друга. Это особенно актуально для веб-приложения, обслуживающего запросы от множества посетителей;

– кроссплатформенность. СУБД может быть развёрнута на любой современной операционной системе: Windows, различные дистрибутивы Linux, macOS;

– активное сообщество и качественная документация. Открытый исходный код и большое количество энтузиастов по всему миру гарантируют быстрое исправление ошибок и доступность огромного объёма справочной информации;

– бесплатное распространение. Отсутствие лицензионных отчислений существенно снижает совокупную стоимость владения системой, что является одним из ключевых требований к проекту.

Решающим фактором в пользу выбора PostgreSQL стала его глубокая и бесшовная интеграция с экосистемой Spring Boot. Фреймворк Spring Data JPA в сочетании с драйвером PostgreSQL позволяет абстрагироваться от написания низкоуровневых SQL-запросов и работать с данными в терминах Java-объектов (ORM), что значительно ускоряет разработку и снижает вероятность ошибок.

СУБД MySQL.

MySQL — ещё одна чрезвычайно популярная реляционная система управления базами данных с открытым исходным кодом. Она широко используется в веб-разработке и является стандартным компонентом многих хостинговых площадок (часто в составе связки LAMP — Linux, Apache, MySQL, PHP).

К достоинствам MySQL относятся:

– простота освоения и администрирования. MySQL имеет интуитивно понятную архитектуру и множество графических утилит для управления (например, phpMyAdmin, MySQL Workbench);

– высокая скорость чтения. Во многих сценариях, ориентированных преимущественно на выборку данных (что характерно для витрины интернет-магазина), MySQL демонстрирует отличную производительность;

– широкая распространённость. Огромное количество веб-сайтов работают под управлением MySQL, что гарантирует наличие большого пула специалистов и готовых решений типовых проблем.

Однако при более детальном сравнении MySQL уступает PostgreSQL в ряде аспектов, важных для разрабатываемого приложения:

– соблюдение стандартов SQL. Исторически MySQL менее строго придерживалась стандарта SQL, допуская некоторые вольности, которые могут привести к неожиданному поведению при миграции на другие СУБД. В

последних версиях ситуация улучшается, но PostgreSQL по-прежнему считается эталоном в этом вопросе;

– поддержка сложных типов данных. Работа с форматом JSON в MySQL появилась позже и реализована менее гибко, чем в PostgreSQL, где тип JSONB позволяет строить индексы для быстрого поиска по содержимому JSON-документов;

– лицензирование. Хотя сама MySQL Community Edition бесплатна, её развитие контролируется корпорацией Oracle, что иногда вызывает опасения у части сообщества относительно будущего проекта.

СУБД SQLite.

SQLite — это компактная встраиваемая реляционная СУБД. Она отличается от клиент-серверных систем (таких как PostgreSQL и MySQL) тем, что вся база данных хранится в одном файле и не требует запуска отдельного серверного процесса. SQLite широко применяется в мобильных приложениях, десктопных программах, встроенных системах и в качестве легковесного хранилища для тестирования.

К преимуществам SQLite можно отнести:

– предельная простота развёртывания. Отсутствует необходимость в установке, настройке и администрировании сервера баз данных — достаточно просто подключить библиотеку к проекту;

– отличная производительность при низкой нагрузке. Для однопользовательских сценариев или небольшого числа одновременных запросов SQLite работает очень быстро;

– портативность. Файл базы данных легко скопировать или перенести на другую машину.

Тем не менее SQLite имеет фундаментальные ограничения, делающие её непригодной для использования в качестве основной СУБД веб-приложения, рассчитанного на обслуживание множества клиентов:

– проблемы с конкурентной записью. SQLite допускает одновременное чтение, но блокирует всю базу данных при выполнении операции записи. В

условиях, когда несколько покупателей могут одновременно оформлять заказы, это приведёт к серьёзному падению производительности и ошибкам;

– ограниченная поддержка типов данных. SQLite использует динамическую типизацию, что может привести к неочевидным ошибкам при работе с данными.

Проанализировав достоинства и недостатки трёх рассмотренных СУБД, для реализации серверной части веб-приложения была выбрана PostgreSQL. Данный выбор обусловлен оптимальным сочетанием таких факторов, как высокая надёжность, полнота поддержки стандарта SQL, развитые возможности по работе с гибкими структурами данных (JSON), отличная производительность при конкурентном доступе и бесшовная интеграция с выбранным фреймворком Spring Boot. Немаловажную роль сыграло и свободное распространение PostgreSQL, что полностью соответствует концепции создания экономически доступного решения для малого бизнеса.

1.2.3 Выбор среды разработки

Для реализации как серверной, так и клиентской частей веб-приложения использовалась единая среда разработки — Visual Studio Code (VS Code). Это бесплатный, кроссплатформенный редактор кода, разработанный компанией Microsoft и получивший широкое распространение среди веб-разработчиков благодаря своей легковесности, высокой производительности и богатой экосистеме расширений.

Выбор Visual Studio Code в качестве основного инструмента разработки был обусловлен его преимуществами. Легковесность и высокая скорость работы. В отличие от полнофункциональных интегрированных сред разработки (IDE), VS Code запускается практически мгновенно и не предъявляет высоких требований к аппаратным ресурсам компьютера. Это позволяет комфортно работать над проектом даже на машинах со средними техническими характеристиками.

Развитая экосистема расширений. Функциональность редактора может быть гибко расширена путём установки дополнений из встроенного маркетплейса. Для работы над проектом были использованы следующие расширения:

– Extension Pack for Java — набор расширений, обеспечивающий полноценную поддержку языка Java: подсветку синтаксиса, автодополнение кода, навигацию по проекту, запуск и отладку приложений на Spring Boot;

– Spring Boot Extension Pack — специализированный набор инструментов для работы с фреймворком Spring Boot, упрощающий управление зависимостями, запуск приложения и навигацию по эндпоинтам контроллеров;

– ESLint и Prettier — инструменты для анализа и автоматического форматирования JavaScript-кода, позволяющие поддерживать единый стиль оформления во всех файлах проекта;

– ES7+ React/Redux/React-Native snippets — расширение, предоставляющее готовые шаблоны кода для быстрого создания компонентов React и других типовых конструкций;

– встроенный терминал. Наличие терминала непосредственно в окне редактора позволяет быстро выполнять команды сборки проекта (mvnw spring-boot:run для серверной части, npm run dev для клиентской части), устанавливать новые зависимости и взаимодействовать с базой данных через утилиту psql, не переключаясь между различными приложениями;

– удобная работа с системой контроля версий Git. VS Code предоставляет наглядный графический интерфейс для выполнения основных операций с Git, а также визуально отображает изменённые, добавленные или удалённые строки кода прямо в редакторе. Это существенно упрощает отслеживание истории изменений и управление версиями проекта;

– кроссплатформенность. Visual Studio Code работает на всех основных операционных системах (Windows, macOS, Linux), что делает его универсальным инструментом, доступным разработчику независимо от используемой платформы.

Таким образом, использование Visual Studio Code в качестве единой среды разработки для обоих компонентов приложения позволило унифицировать рабочий процесс, снизить требования к аппаратному обеспечению.

1.2.4 Выбор графической библиотеки

Для создания клиентской части интернет-магазина нужно было выбрать библиотеку, которая позволит построить живой, отзывчивый и удобный в поддержке интерфейс. Рассматривали три самых популярных на сегодня решения: React, Vue.js и Angular.

Библиотека React.

React — это декларативная JavaScript-библиотека с открытым исходным кодом, которую разрабатывает и поддерживает компания Meta. Она создана для построения интерактивных интерфейсов на основе компонентного подхода. React не является фреймворком в полном смысле слова — он сосредоточен именно на отображении данных, и в этом его гибкость: его легко комбинировать с другими библиотеками и инструментами.

Что можно отнести к сильным сторонам React:

– компонентный подход. Интерфейс собирается из независимых, переиспользуемых компонентов. Каждый компонент содержит в себе разметку, стили и логику поведения. Такой подход заметно упрощает разработку, тестирование и дальнейшее сопровождение. Можно спокойно менять одну часть системы, не опасаясь, что что-то сломается в другой;

– виртуальный DOM. React работает не с браузерной объектной моделью напрямую, а с её лёгким представлением в памяти. Когда состояние компонента меняется, React вычисляет, какие именно изменения нужно внести в реальный DOM, и применяет их пакетно. Это даёт хорошую производительность и плавную работу интерфейса, даже когда происходит частое обновление данных;

– однонаправленный поток данных. Данные передаются от родительских компонентов к дочерним через свойства. Такая архитектура делает поведение приложения более предсказуемым и упрощает отладку — всегда понятно, откуда пришли те или иные данные;

– широкая экосистема и активное сообщество. Вокруг React сложилось огромное сообщество разработчиков. Под любую задачу есть готовая библиотека: маршрутизация — React Router, управление состоянием — Redux или Zustand, работа с формами — React Hook Form, HTTP-запросы — Axios. Выбор действительно большой;

– поддержка современного синтаксиса. React легко интегрируется с современными инструментами сборки вроде Vite или Webpack, что позволяет использовать все новые возможности JavaScript, включая хуки. Они дают удобный и лаконичный способ управлять состоянием и побочными эффектами прямо в функциональных компонентах.

Фреймворк Vue.js.

Vue.js — это прогрессивный JavaScript-фреймворк для создания интерфейсов, который разработал Эван Ю. Его главные отличия — низкий порог входа и плавная кривая обучения. Для начинающих разработчиков и небольших проектов это весьма привлекательный вариант.

Сильные стороны Vue.js:

– простота и интуитивная понятность. Синтаксис близок к классическим HTML, CSS и JavaScript — можно начинать работать практически сразу, без долгого погружения в новую парадигму;

– отличная документация. Официальное руководство Vue.js считается одним из лучших в индустрии. Ответы на большинство вопросов находятся быстро;

– гибкость. Vue.js можно встраивать в уже существующий проект постепенно, начав буквально с пары компонентов на странице.

При всех плюсах Vue.js уступает React по размеру экосистемы и по количеству вакансий на рынке. Для выпускной квалификационной работы этот момент тоже имеет значение — технология, более востребованная работодателями, выглядит предпочтительнее.

Фреймворк Angular.

Angular — это полнофункциональный фреймворк от Google, рассчитанный на крупные и сложные веб-приложения. В отличие от React и Vue, он даёт готовые решения «из коробки» практически для всех типовых задач: маршрутизация, формы, HTTP-запросы, управление состоянием.

Что можно записать в плюсы Angular:

– строгая и продуманная архитектура. Angular задаёт чёткие правила организации кода, благодаря чему проект получается единообразным, а новым разработчикам проще погружаться в проект;

– TypeScript. Использование строго типизированного языка помогает отлавливать многие ошибки ещё на этапе компиляции, что особенно ценно в больших проектах;

– богатый встроенный функционал. Не нужно подбирать и настраивать сторонние библиотеки для стандартных задач — всё уже есть.

Однако у Angular есть и серьёзные минусы, которые делают его менее подходящим для данной работы:

– высокий порог входа. На освоение Angular уходит ощутимо больше времени и сил, чем на React или Vue;

– избыточность для небольших проектов. Многие возможности фреймворка при разработке сравнительно небольшого интернет-магазина просто не понадобятся, но добавят сложности и увеличат итоговый размер приложения.

Взвесив все плюсы и минусы трёх вариантов, для клиентской части был выбран React. В этом решении сложились в удачную пропорцию сразу несколько важных вещей: высокая производительность, гибкость, огромная экосистема готовых библиотек и компонентов, а также реальная востребованность технологии на рынке. Компонентный подход и механизм виртуального DOM позволили сделать современный, быстрый и удобный в сопровождении интерфейс, который полностью отвечает всем требованиям, поставленным перед интернет-магазином спортивных товаров.

1.2.5 Обмен данными JSON

Для организации взаимодействия между клиентской и серверной частями разрабатываемого веб-приложения требовался универсальный, легковесный и удобный в обработке формат передачи данных. В качестве такого формата был выбран JSON (JavaScript Object Notation).

JSON — это стандартный текстовый формат, предназначенный для хранения и обмена структурированными данными. Он основан на синтаксисе описания объектов в языке JavaScript, однако является полностью независимым от него. Благодаря своей простоте и выразительности JSON получил широчайшее распространение и поддерживается практически всеми современными языками программирования, включая Java (серверная часть проекта) и JavaScript (клиентская часть).

Выбор JSON в качестве основного формата обмена данными между компонентами приложения был обусловлен рядом его существенных преимуществ:

– простота и читаемость синтаксиса. Данные в формате JSON представляются в виде пар «ключ — значение» и упорядоченных списков, что интуитивно понятно как человеку, так и компьютеру. Это упрощает отладку приложения, поскольку содержимое HTTP-запросов и ответов можно легко просмотреть и проанализировать в инструментах разработчика браузера;

– компактность и высокая скорость передачи. JSON-представление данных содержит минимум служебной информации по сравнению, например, с форматом XML. Меньший объём передаваемых данных способствует ускорению загрузки страниц и снижению нагрузки на сеть, что особенно важно для пользователей с медленным или нестабильным интернет-соединением;

– широкая языковая поддержка. Для работы с JSON существуют готовые и хорошо оптимизированные библиотеки для всех популярных языков программирования. В серверной части проекта (Spring Boot) сериализация Java-объектов в JSON и обратное преобразование выполняются автоматически с помощью встроенной библиотеки Jackson. В клиентской части (React) объекты JSON являются нативными для JavaScript и обрабатываются стандартными средствами языка;

– гибкость структуры. JSON позволяет описывать данные любой степени вложенности: простые типы (строки, числа, логические значения), массивы и вложенные объекты. Это делает его идеальным форматом для передачи сложных структур данных, таких как карточка товара с набором характеристик, состав заказа с перечнем позиций или конфигурация фильтров для страницы каталога;

– возможность включения метаданных. При необходимости в JSON-документ могут быть добавлены дополнительные поля, описывающие представленные данные (например, общее количество записей, номер текущей страницы, статус выполнения запроса), что упрощает реализацию постраничной навигации и обработку ошибок.

В разрабатываемом веб-приложении JSON используется для всех видов взаимодействия между клиентом и сервером. Клиентское приложение на React отправляет на сервер HTTP-запросы (GET, POST), передавая параметры фильтрации и сортировки, а также данные оформленного заказа. Серверное приложение на Spring Boot обрабатывает поступившие запросы, извлекает необходимую информацию из базы данных, формирует ответ в виде JSON-документа и отправляет его обратно клиенту. Полученный ответ парсится и отображается в пользовательском интерфейсе.

Таким образом, использование формата JSON обеспечило эффективный, стандартизированный и легко отлаживаемый обмен данными между компонентами распределённого веб-приложения, что полностью соответствует современным подходам к разработке клиент-серверных систем.

1.3 Проектирование информационной системы

1.3.1 Паттерн MVC

Архитектура веб-приложения интернет-магазина спортивных товаров построена по широко известному шаблону проектирования MVC — Model, View, Controller. Этот подход предполагает разделение системы на три основных компонента, каждый из которых можно дорабатывать и менять относительно независимо от остальных. Для разработки, тестирования и дальнейшего сопровождения продукта это даёт заметные преимущества.

Рассмотрим, какую роль играет каждый из компонентов в рамках данной системы.

Модель. Модель содержит бизнес-логику приложения и отвечает за всё, что связано с данными. В серверной части модель представлена классами-сущностями, которые отображают структуру таблиц базы данных — например, Ball, Shoes, Order. К ним примыкают классы-репозитории, через которые выполняются операции чтения и записи. Модель получает команды от контроллера, при необходимости обращается к базе, выполняет вычисления — скажем, фильтрацию или сортировку товаров — и возвращает готовые данные обратно контроллеру.

Представление. Представление отвечает за то, что видит пользователь, и за интерфейс, с которым он взаимодействует. В разработанном приложении роль представления выполняет клиентская часть, написанная на React. Каждая страница — каталог, карточка товара, корзина — собрана из React-компонентов. Они получают данные от сервера через REST API и превращают их в HTML-разметку. Представление также обрабатывает первичные действия посетителя — нажатия на кнопки, выбор фильтров — и передаёт сведения о них дальше, на уровень логики.

Контроллер. Контроллер связывает модель и представление. В серверной части контроллеры реализованы в виде классов с аннотациями @RestController и @RequestMapping. Каждый метод такого класса привязан к определённому HTTP-

эндпоинту — например, /api/ball или /api/page/{id}. Контроллер принимает запрос от клиента, извлекает из него переданные параметры — фильтры, номер страницы, сортировку, — вызывает нужные методы модели и, получив от неё данные, формирует ответ в формате JSON и отправляет клиенту. Помимо этого, контроллер занимается проверкой входящих данных и обработкой ошибок.

В современной веб-разработке нередко применяют и другой шаблон — MVVM (Model, View, ViewModel), особенно популярный во фреймворках Vue.js и Angular. В этой схеме:

– модель хранит данные и бизнес-логику;

– представление показывает данные и реагирует на действия пользователя;

– ViewModel служит прослойкой между ними: обрабатывает логику отображения и управляет данными, которые выводятся в представлении.

Хотя библиотека React, использованная в клиентской части, концептуально тяготеет к MVVM — компоненты React вполне можно рассматривать как ViewModel, — архитектура всего приложения в целом выдержана в классической схеме MVC. Такой выбор сделан по нескольким причинам:

– чёткое разделение обязанностей. Контроллеры Spring Boot полностью берут на себя обработку HTTP-запросов, модель работает только с данными, а React-приложение занимается исключительно отображением. Код получается организованным, его легче поддерживать;

– удобное управление запросами к базе. Контроллеры Spring MVC дают развитые средства для извлечения параметров запроса, проверки данных и обработки ошибок — а для клиент-серверного приложения, которое активно взаимодействует с базой, это особенно важно;

– соответствие отраслевому стандарту. Связка Spring Boot на сервере и React на клиенте — одно из самых распространённых и хорошо описанных архитектурных решений для современных веб-приложений.

Применение паттерна MVC позволило получить хорошо структурированную, масштабируемую и удобную в сопровождении архитектуру,

которая полностью отвечает требованиям, заложенным в проект интернет-магазина.

1.3.2 Архитектура приложения

На рисунке 4 представлена архитектура разработанного веб-приложения интернет-магазина спортивных товаров.

В процессе работы с интернет-магазином пользователь взаимодействует с графическим интерфейсом, отображаемым в браузере. Каждая страница приложения (каталог товаров, карточка товара, корзина) формируется набором React-компонентов, которые в совокупности образуют уровень представления (View). Например, при оформлении заказа пользователь заполняет форму, вводя контактный номер телефона, и инициирует отправку данных нажатием соответствующей кнопки.

Когда пользователь нажимает кнопку оформления заказа, клиентский код на JavaScript формирует HTTP-запрос и отправляет его на сервер. На стороне сервера этот запрос поступает в компонент группы Controller (контроллер Spring Boot), который выполняет первичную проверку и валидацию поступивших данных. Если номер телефона соответствует требуемому формату, управление передаётся компоненту Model (сервисный слой и репозитории Spring Data JPA), задача которого — сохранить заказ в базе данных PostgreSQL и вернуть результат операции. После успешного сохранения контроллер формирует ответ и отправляет его обратно клиенту, который, в свою очередь, отображает пользователю уведомление об успешном оформлении заказа. В случае, если валидация данных на стороне сервера не пройдена, контроллер возвращает сообщение об ошибке, которое интерпретируется React-компонентом и выводится пользователю в соответствующем виде.

Вся работа с приложением будет осуществляться в виде диалога с пользователем. Диалоговая система состоит из трех ключевых компонентов

– клиентская часть, которая обеспечивает пользовательский интерфейс для взаимодействия с системой, включая кнопки, поля ввода, выпадающие списки, таблицы, чекбоксы и др.;

– серверная часть, которая обрабатывает данные, отправленные пользователем, выполняет необходимые алгоритмы и задачи;

– база данных, где хранятся данные с определенными правилами и принципами управления данными.

Система управления базами данных (СУБД) представляет собой набор программных инструментов, которые позволяют создавать, управлять и администрировать базы данных. Другими словами, СУБД предоставляет возможности для организации, контроля и администрирования данных в базах данных.

На рисунке 6 представлено взаимодействие компонентов приложения в виде диаграммы последовательности.

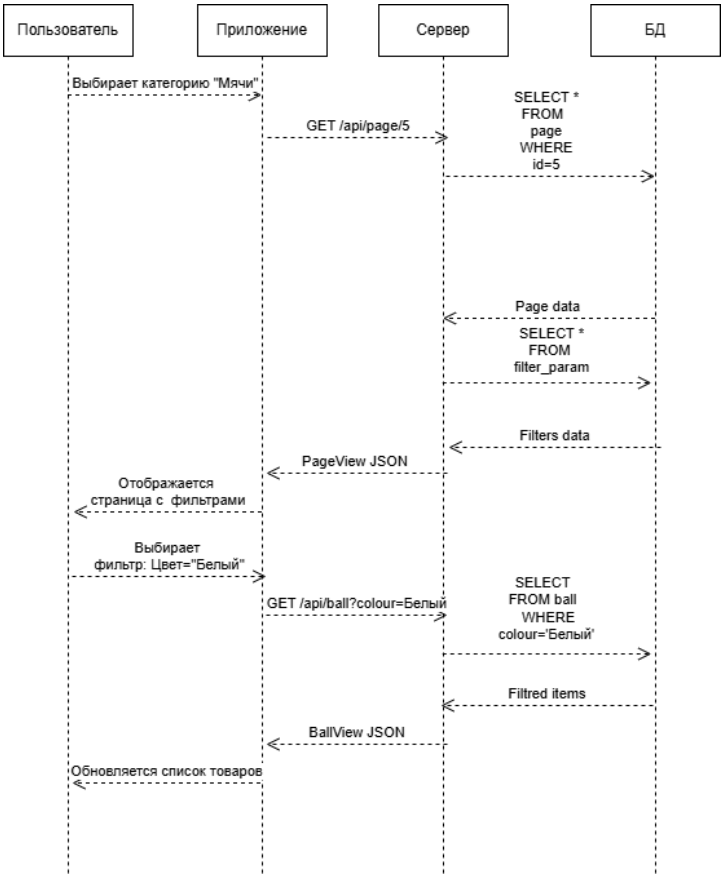


Рисунок 6 – Диаграмма взаимодействия

Диаграмма классов, представляющая модель данных, изображена на рисунке 7.

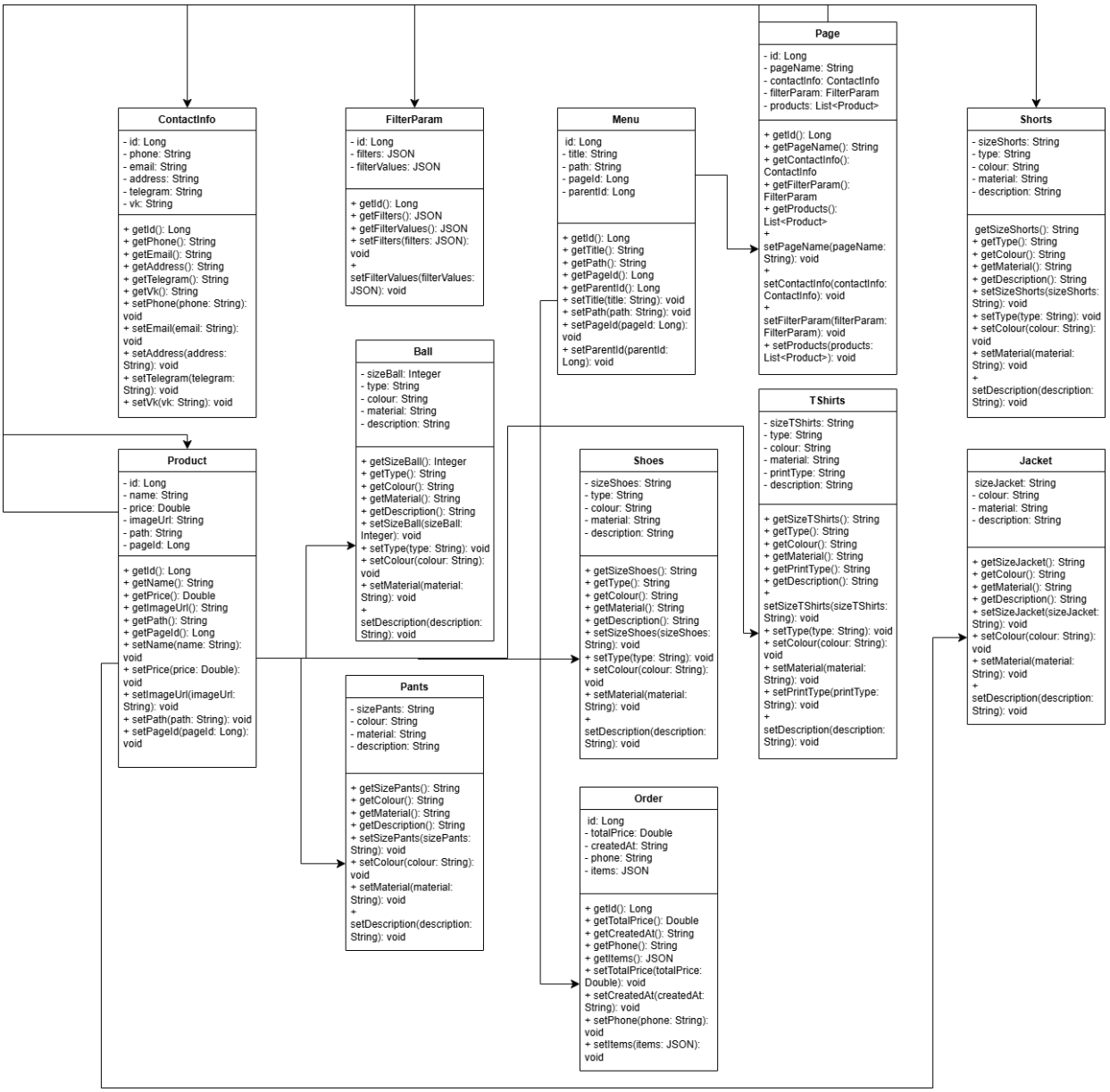


Рисунок 7 – Диаграмма классов

На рисунке 8 изображена диаграмма деятельности.

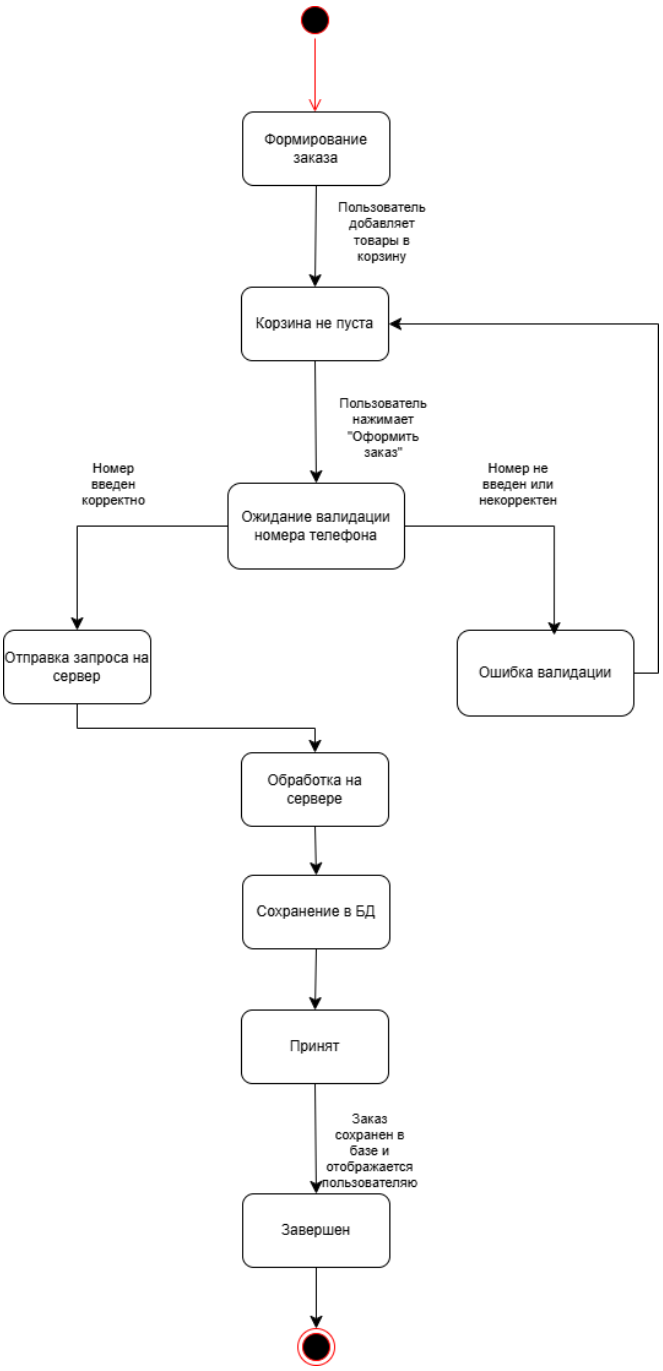


Рисунок 8 – Диаграмма состояний для оформление заказа

1.3.3 Проектирование базы данных

Для хранения и управления всей информацией, необходимой для функционирования интернет-магазина спортивных товаров, была спроектирована реляционная база данных под управлением СУБД PostgreSQL. Структура базы данных разрабатывалась с учётом требований нормализации, обеспечения целостности данных и эффективности выполнения типовых запросов.

В составе базы данных были созданы следующие таблицы:

– page — таблица, содержащая информацию о страницах каталога;

– contact_info — таблица, хранящая контактные данные магазина (телефон, email, адрес, социальные сети);

– filter_param — таблица, содержащая конфигурацию фильтров для каждой страницы каталога (перечень параметров и их возможные значения в формате JSON);

– menu — таблица, описывающая структуру бокового меню интернет-магазина;

– product — таблица, представляющая собой витрину товаров на главной странице и страницах-категориях верхнего уровня;

– ball — таблица, хранящая детальную информацию о товарах категории «Мячи»;

– shoes — таблица, содержащая информацию о товарах категории «Обувь»;

– tshirts — таблица, хранящая данные о товарах категории «Футболки»;

– shorts — таблица, содержащая информацию о товарах категории «Шорты»;

– jacket — таблица, хранящая данные о товарах категории «Куртки и ветровки»;

– pants — таблица, содержащая информацию о товарах категории «Брюки»;

– sweater — таблица, хранящая данные о товарах категории «Джемперы, олимпийки, толстовки, худи»;

– sports_underwear — таблица, содержащая информацию о товарах категории «Спортивное бельё»;

– protective_cloth — таблица, хранящая данные о товарах категории «Защитная экипировка»;

– gloves — таблица, содержащая информацию о товарах категории «Вратарские перчатки»;

– gaiters — таблица, хранящая данные о товарах категории «Гетры и гольфы»;

– accessories — таблица, содержащая информацию о товарах категории «Аксессуары»;

– inventory — таблица, хранящая данные о товарах категории «Инвентарь»;

– orders — таблица, предназначенная для хранения информации об оформленных заказах.

Для обеспечения ссылочной целостности между таблицами определены связи по внешним ключам. Таблица page ссылается на contact_info и filter_param, таблица menu ссылается на page, а таблица product связана с page. Таблицы конкретных категорий товаров (ball, shoes и другие) являются независимыми и содержат полную информацию о соответствующих товарных позициях.

Физическая модель базы данных в виде ER-диаграммы, на которой детально представлены все таблицы, их атрибуты и связи между ними, приведена на рисунке 9.

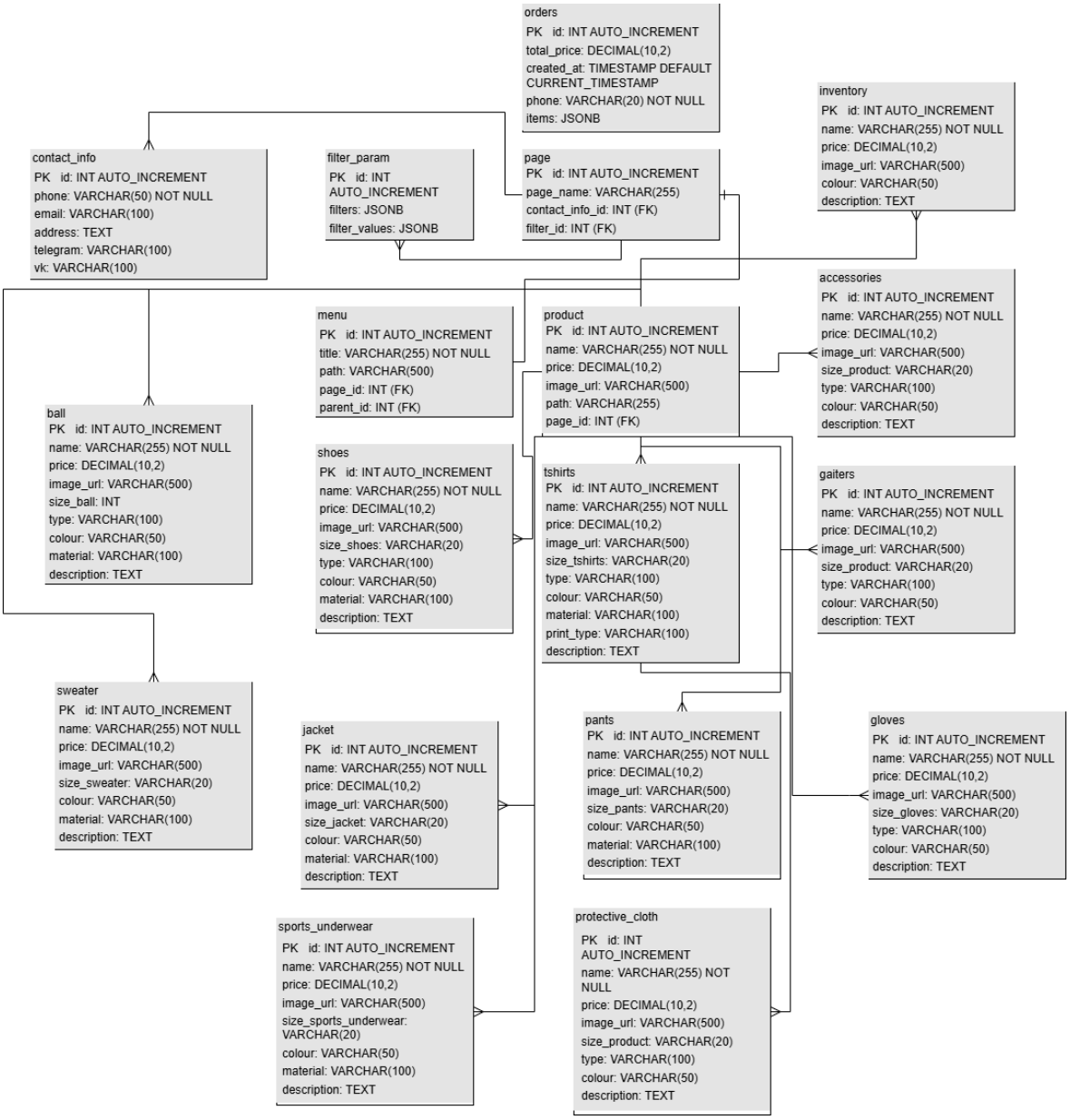


Рисунок 9 – ER диаграмма для БД

1.4 Разработка приложения

1.4.1 Реализация базы данных

Для реализации базы данных интернет-магазина была выбрана система управления базами данных PostgreSQL. В отличие от ручного выполнения SQL-скриптов, для управления структурой базы данных (создание таблиц, связей и индексов) в проекте используется инструмент миграций Flyway. Такой подход позволяет автоматически разворачивать базу данных нужной структуры на любой машине, где запускается приложение, и вести историю изменений схемы данных.

При первом запуске серверной части на Spring Boot инструмент Flyway автоматически сканирует папку src/main/resources/db/migration, находит там файлы миграций (специальные SQL-скрипты с определёнными именами) и последовательно применяет их. Этот процесс гарантирует, что база данных всегда находится в актуальном и согласованном состоянии.

В рамках проекта были разработаны следующие ключевые миграции.

Первая миграция, V1_create_page_tables.sql, отвечает за создание основных таблиц, формирующих структуру каталога и контент страниц. Пример создания таблицы page, содержащей информацию о страницах каталога, представлен ниже:

```
CREATE TABLE page (
    id BIGSERIAL PRIMARY KEY,
    page_name  VARCHAR(255),
    contact_info_id  BIGINT,
    filter_id BIGINT
);
```

Для хранения контактной информации магазина была создана таблица contact_info. SQL-запрос для её создания выглядит следующим образом:

```
CREATE TABLE contact_info (
    id BIGSERIAL PRIMARY KEY,
    phone VARCHAR(50),
    email VARCHAR(100),
    address TEXT,
    telegram VARCHAR(100),
```

```
    vk VARCHAR(100)
);
```

Для хранения конфигурации фильтров, специфичных для каждой категории товаров, была спроектирована таблица filter_param. Она использует тип данных JSONB для гибкого хранения возможных значений фильтров:

```
CREATE TABLE filter_param (
    id BIGSERIAL PRIMARY KEY,
    filters JSONB,
    filter_values JSONB
);
```

Вторая миграция, V2_insert_page_tables.sql, отвечает за наполнение базы данных первоначальными данными, необходимыми для демонстрации работы приложения. На этом этапе в таблицы вставляются тестовые товары, пункты меню, структура страниц и значения фильтров.

Пример добавления нового товара в категорию «Мячи» (таблица ball) выглядит следующим образом:

```
INSERT INTO ball (name, price, image_url, size_ball, type,
colour, description, material)
VALUES ('Мяч футбольный Meridian Elite 5', 2690, 'myach-
futbolnyy-meridian-elite-5.png',
            5, 'футбольный', 'белый',
            'Футбольный мяч Meridian Elite. Используется для
тренировок и любительских матчей. Сбалансированный и предсказуемый в
полете.',
            'ПВХ');
```

Пример добавления нового пункта в боковое меню сайта (таблица menu) выглядит так:

```
INSERT INTO menu (title, path, page_id, parent_id)
VALUES ('Мячи', '/api/page/5', 5, NULL);
```

Пример вставки значений фильтров для категории «Мячи» (таблица filter_param):

```
INSERT INTO filter_param (filters, filter_values)
VALUES  ('["colour","type","sizeBall","material"]',
```

```
            '{"sizeBall": ["3", "4", "5"], "type": ["футбольный",
"баскетбольный"],
              "colour": ["белый", "оранжевый"], "material": ["ПВХ",
"резина"]}');
```

Для добавления контактной информации магазина в таблицу contact_info использовался следующий запрос:

```
INSERT INTO contact_info (phone, email, address, telegram, vk)
VALUES ('+7 999 123-45-67', 'info@sportswear.ru', 'г. Пенза,
ул. Красная, д. 1',
          '@sportswear_shop',  'vk.com/sportswear_shop');
```

Все вышеперечисленные операции по созданию и наполнению базы данных выполняются автоматически при запуске приложения, что исключает необходимость ручного вмешательства и минимизирует риск возникновения ошибок, связанных с человеческим фактором.

1.4.2 Описание программы

Описание клиентской части приложения.

Клиентская часть веб-приложения интернет-магазина представляет собой одностраничное приложение (SPA), построенное с использованием библиотеки React. Исходный код клиентской части организован в виде набора директорий и файлов, каждый из которых выполняет определённую функциональную роль.

В корневой папке проекта располагаются конфигурационные файлы, определяющие параметры сборки приложения и его зависимости. Файл package.json содержит перечень всех используемых библиотек, таких как react, react-router-dom (для маршрутизации) и axios (для выполнения HTTP-запросов). Инструмент сборки Vite обеспечивает быстрый запуск проекта в режиме разработки и создание оптимизированной производственной сборки.

Файл .env, также расположенный в корне проекта, хранит переменные окружения, используемые для настройки приложения в зависимости от среды выполнения. В частности, в нём задаётся базовый URL-адрес серверного API:

```
VITE_API_BASE_URL=http://localhost:8080
```

В папке src располагается весь исходный код, написанный на языке JavaScript с использованием расширения синтаксиса JSX. Структура папки src организована следующим образом:

– папка api содержит модули, отвечающие за взаимодействие с серверной частью приложения;

– папка components хранит переиспользуемые компоненты пользовательского интерфейса, такие как модальные окна и индикаторы загрузки;

– папка context содержит определения контекстов React для управления глобальным состоянием приложения;

– папка pages включает компоненты, представляющие собой целые страницы приложения;

– папка shared содержит вспомогательные функции, используемые в различных частях приложения.

В папке api находится файл client.js, который создаёт настроенный экземпляр HTTP-клиента Axios. В нём задаётся базовый URL, к которому будут обращаться все запросы, и устанавливается тайм-аут ожидания ответа:

```
import axios from 'axios'
const API_BASE_URL = import.meta.env.VITE_API_BASE_URL || '/'
export const api = axios.create({
  baseURL: API_BASE_URL,
  timeout: 10000,
})
```

Файл shopApi.js, также расположенный в папке api, экспортирует набор функций, каждая из которых соответствует определённому эндпоинту серверного REST API. Эти функции используют экземпляр api для выполнения HTTP-запросов и возвращают полученные данные. Пример функции для получения списка мячей с учётом параметров фильтрации и пагинации:

```
export function getBalls(params) {
  return  requestList('/api/ball',  params)
}
```

Функция requestList, используемая внутри shopApi.js, добавляет переданные параметры к URL запроса:

```
function requestList(endpoint, params) {
  const hasParams = params && Object.keys(params).length > 0
  return requestData(() => api.get(endpoint, hasParams ? { params } : undefined))
}
```

В папке pages располагаются компоненты, отвечающие за отрисовку целых страниц приложения. Одним из ключевых является компонент PageItems.jsx, который реализует логику отображения каталога товаров для конкретной категории. Внутри этого компонента определяются следующие функции и состояния:

– функция pageRequestById, устанавливающая соответствие между идентификатором страницы и функцией API для загрузки товаров соответствующей категории;

– функция normalizeByPage, приводящая данные, полученные от сервера, к единому формату для отображения в карточке товара;

– состояние items, хранящее массив товаров, загруженных для текущей страницы;

– состояние filters, хранящее выбранные пользователем значения фильтров и сортировки.

Упрощённый фрагмент компонента PageItems.jsx, демонстрирующий загрузку данных при изменении параметров фильтрации:

```
useEffect(() => {
  if (!requestFn || requestResolvedForPageId !== pageId) return
  loadItems(buildParamsFromFilters(filters))
}, [requestFn, filters, currentPage, buildParamsFromFilters, loadItems, requestResolvedForPageId, pageId])
```

Для управления состоянием корзины покупок в приложении используется React Context API. Определение контекста и провайдера находится в файле CartContext.jsx, расположенном в папке context. Провайдер CartProvider

оборачивает всё приложение и предоставляет дочерним компонентам доступ к текущему содержимому корзины и функциям для его изменения:

– addItem — функция для добавления нового товара в корзину;

– removeItem — функция для удаления товара из корзины по его идентификатору;

– clearCart — функция для полной очистки корзины;

– totalItems — количество позиций в корзине;

– totalPrice — общая стоимость всех товаров в корзине.

Данные корзины сохраняются в локальном хранилище браузера (localStorage), что позволяет восстанавливать её содержимое после закрытия и повторного открытия страницы.

Функция для добавления товара в корзину выглядит следующим образом:

```
const addItem = (product) => {
  const normalized = normalizeCartProduct(product)
  setItems((prev) => [...prev, { ...normalized, cartItemId: makeCartItemId()  }])
}
```

Компонент CartDrawer.jsx, расположенный в папке components, отвечает за отображение выдвижной панели корзины. Он получает доступ к данным и функциям корзины через хук useCart и отрисовывает список добавленных товаров, итоговую сумму и форму для ввода номера телефона. При нажатии кнопки «Оформить заказ» компонент вызывает функцию createOrder из модуля shopApi.js, передавая ей состав заказа и контактный номер телефона.

Вспомогательные компоненты, такие как LoadingState (индикатор загрузки) и ErrorState (отображение ошибки), используются для улучшения пользовательского опыта и информирования посетителя о текущем состоянии приложения.

Таким образом, клиентская часть приложения реализована в виде набора независимых, переиспользуемых компонентов, взаимодействующих между собой через чётко определённые интерфейсы (props и контекст). Такая архитектура

обеспечивает высокую скорость работы интерфейса, удобство его сопровождения и возможность дальнейшего расширения функциональности.Описание серверной части приложения

Серверная часть веб-приложения интернет-магазина реализована на языке Java с использованием фреймворка Spring Boot. Серверное приложение построено по многослойной архитектуре, включающей в себя уровень контроллеров (Controllers), уровень сервисов (Services) и уровень доступа к данным (Repositories). Взаимодействие с базой данных PostgreSQL осуществляется посредством технологии Spring Data JPA, а управление структурой базы данных возложено на инструмент миграций Flyway.В корне проекта располагается основной класс приложения, аннотированный @SpringBootApplication. Он содержит единственный метод main, который делегирует запуск всего приложения встроенному механизму Spring Boot:

```
@SpringBootApplication
public class SportswearApplication {
    public static void main(String[] args) {
        SpringApplication.run(SportswearApplication.class,
args);
    }
}
```

Запуск этого класса приводит к старту встроенного веб-сервера Apache Tomcat, который начинает прослушивать порт 8080 (определён в файле конфигурации application.properties) и ожидать входящие HTTP-запросы.

В пакете sportswear.sportswear.config располагаются конфигурационные классы, настраивающие различные аспекты работы приложения. Одним из важнейших является класс WebConfig, который отвечает за глобальную настройку политики CORS (Cross-Origin Resource Sharing). Благодаря ему клиентское приложение, работающее на другом порту (5173), может беспрепятственно отправлять запросы к серверному API:

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
                .allowedOrigins("http://localhost:5173")
                .allowedMethods("GET", "POST", "PUT", "DELETE",
"OPTIONS");
    }
}
```

Уровень контроллеров представлен набором классов, размеченных аннотацией @RestController. Каждый такой класс отвечает за обработку запросов, относящихся к определённой группе эндпоинтов. Например, класс BallController обрабатывает запросы, связанные с категорией товаров «Мячи». Он содержит методы, соответствующие HTTP-глаголам GET и определённым URL-шаблонам:

```
@RestController
@RequestMapping("/api")
public class BallController {
    private final BallService ballService;

    @GetMapping("/ball")
    public BallView getBalls(
            @RequestParam(defaultValue = "10") int size,
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(required = false) Integer sizeBall,
            @RequestParam(required = false) String type,
            @RequestParam(required = false) String colour,
            @RequestParam(required = false) String sortBy,
            @RequestParam(required = false) String direction) {
        return ballService.getBalls(page, size, sizeBall, type,
colour, sortBy, direction);
    }

    @GetMapping("/ball/{id}")
```

```
    public Ball getBallById(@PathVariable Long id) {
        return ballService.getBallById(id);
    }
}
```

Метод getBalls принимает набор параметров, соответствующих возможностям фильтрации и пагинации, и делегирует выполнение бизнес-логики сервисному слою. Аналогичные контроллеры реализованы для всех остальных категорий товаров: ShoesController, TShirtsController, ShortsController и других.

Отдельного внимания заслуживает класс PageController, отвечающий за предоставление данных о структуре конкретной страницы каталога:

```
@RestController
@RequestMapping("/api")
public class PageController {
    private final PageService pageService;

    @GetMapping("/page/{id}")
    public PageView getPageById(@PathVariable Long id) {
        return pageService.getPageById(id);
    }
}
```

Метод getPageById возвращает объект, содержащий не только базовую информацию о странице, но и связанные с ней товары, контактную информацию и конфигурацию фильтров, что позволяет клиентскому приложению динамически настраивать интерфейс.

Уровень сервисов (классы с аннотацией @Service) инкапсулирует в себе всю бизнес-логику приложения. Например, класс BallService отвечает за обработку параметров фильтрации и сортировки, преобразование их в объекты спецификаций Spring Data JPA и выполнение запросов к базе данных через соответствующий репозиторий:

```
@Service
@AllArgsConstructor
public class BallService {
    private final BallRepository ballRepository;

    public BallView getBalls(int page, int size, Integer
sizeBall, String type,
                             String colour, String sortBy,
String direction) {
        Pageable pageable = PageRequest.of(page, size,
buildSort(sortBy, direction));
        Specification<Ball> spec = buildSpecification(sizeBall,
type, colour);
        Page<Ball> ballPage = ballRepository.findAll(spec,
pageable);
        BallView view = new BallView();
        view.setItems(ballPage.getContent());
        view.setCount(ballPage.getTotalElements());
        return view;
    }

    public Ball getBallById(Long id) {
        return ballRepository.findById(id).orElse(null);
    }
}
```

Сервисный слой взаимодействует с уровнем доступа к данным, который представлен интерфейсами-репозиториями. Например, интерфейс BallRepository наследуется от JpaRepository и JpaSpecificationExecutor, что автоматически предоставляет набор стандартных методов для выполнения операций чтения и записи, а также возможность динамического построения запросов с использованием спецификаций:

```
    @Repository
    public interface BallRepository extends JpaRepository<Ball,
Long>,

JpaSpecificationExecutor<Ball>  {
    }
```

Для управления структурой базы данных в проекте используется Flyway. Все миграции (SQL-скрипты) хранятся в папке src/main/resources/db/migration. При каждом запуске приложения Flyway автоматически проверяет, какие миграции ещё не были применены к базе данных, и последовательно выполняет их. Это гарантирует, что схема данных всегда находится в актуальном состоянии и соответствует ожидаемой структуре.

Сериализация Java-объектов в формат JSON и обратное преобразование выполняются автоматически с помощью встроенной в Spring Boot библиотеки Jackson. Благодаря этому клиентское приложение получает данные в удобном для обработки формате, а сервер, в свою очередь, может принимать JSON-представления объектов (например, при оформлении заказа) и автоматически преобразовывать их в экземпляры Java-классов.

Таким образом, серверная часть приложения реализует все необходимые функции для обработки запросов, поступающих от клиента, выполнения бизнес-логики, взаимодействия с базой данных и возврата результатов в стандартизированном формате JSON. Многослойная архитектура обеспечивает чёткое разделение ответственности между компонентами, что упрощает тестирование, отладку и дальнейшее развитие системы.

1.4.3 Руководство пользователя

Чтобы начать работу с интернет-магазином, пользователю достаточно открыть веб-браузер и перейти по адресу, на котором развёрнуто приложение. После загрузки стартовой страницы (рисунок 11) посетитель попадает на главную страницу каталога, где представлены основные категории товаров и витрина с популярными позициями.

В верхней части страницы располагается навигационное меню, содержащее ссылки на разделы каталога: «Обувь», «Одежда», «Мячи», «Аксессуары», «Инвентарь» и другие. В нижней части страницы (футере) отображается контактная информация магазина: телефон, адрес электронной почты, физический адрес и ссылки на социальные сети (рисунок 12).

Для просмотра товаров определённой категории достаточно выбрать соответствующий пункт в боковом меню, после чего откроется страница каталога с возможностью фильтрации и сортировки товаров.

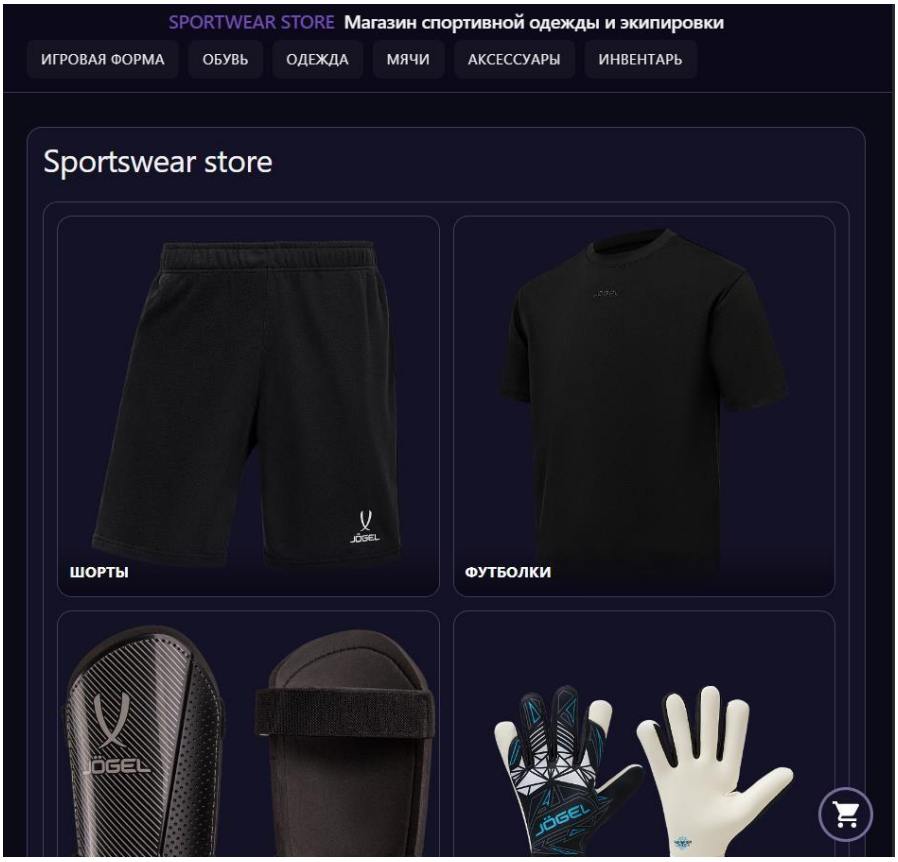


Рисунок 11 – Главная страница

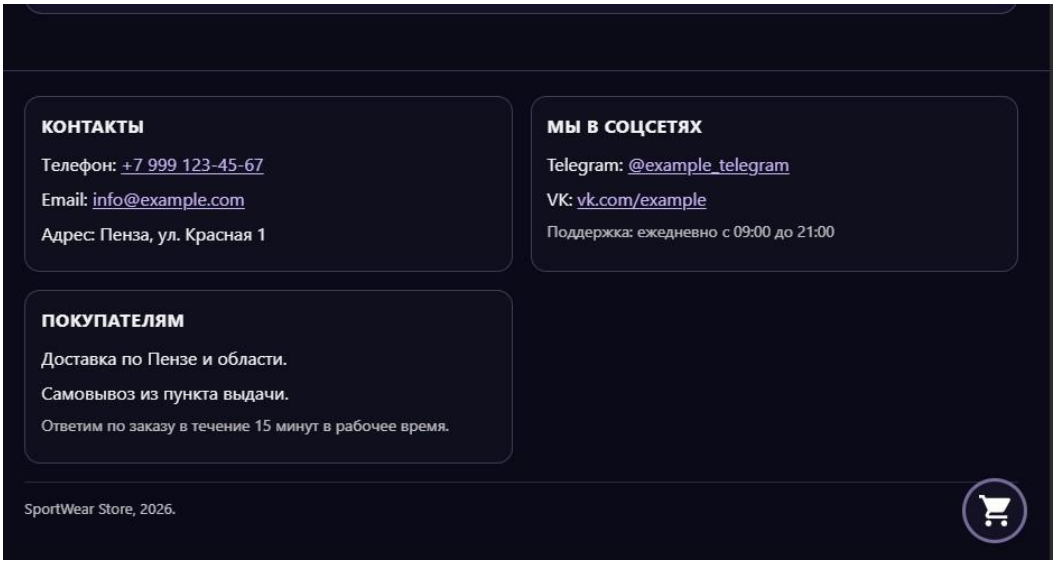


Рисунок 12 – Нижняя часть сайта

При нажатии на одну из кнопок в верхней части сайта можно отсортировать товар по категориям (рисунок 13).

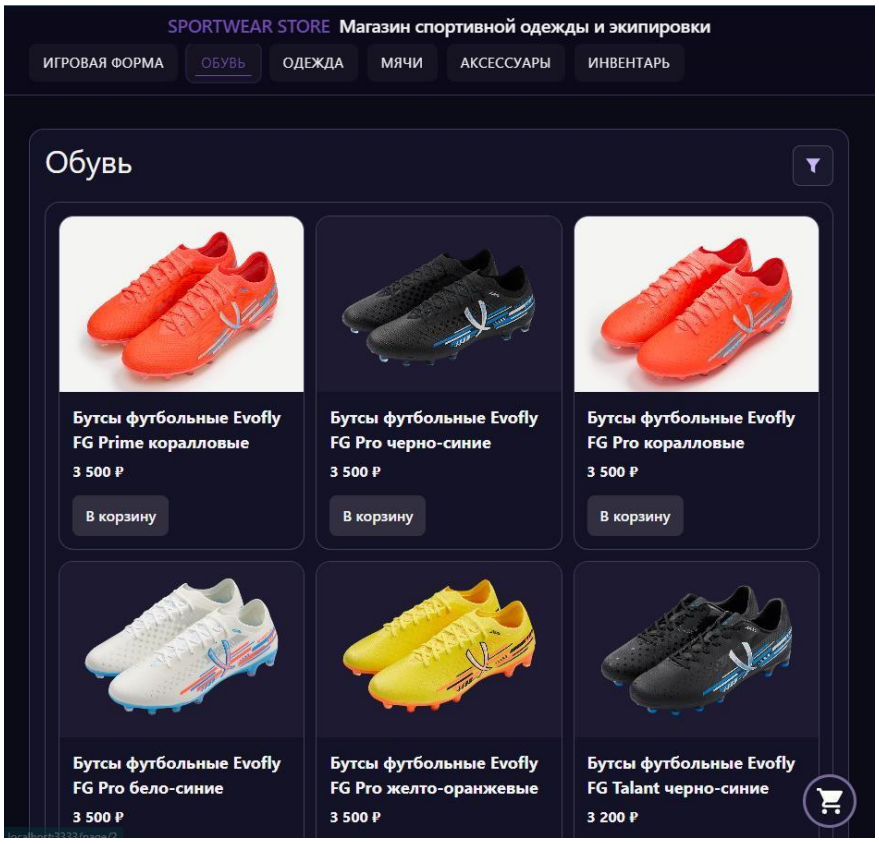


Рисунок 13 – Сортировка

При нажатии на кнопку фильтрации можно отфильтровать товар по цвету, типу, размеру и тд (рисунок 14).

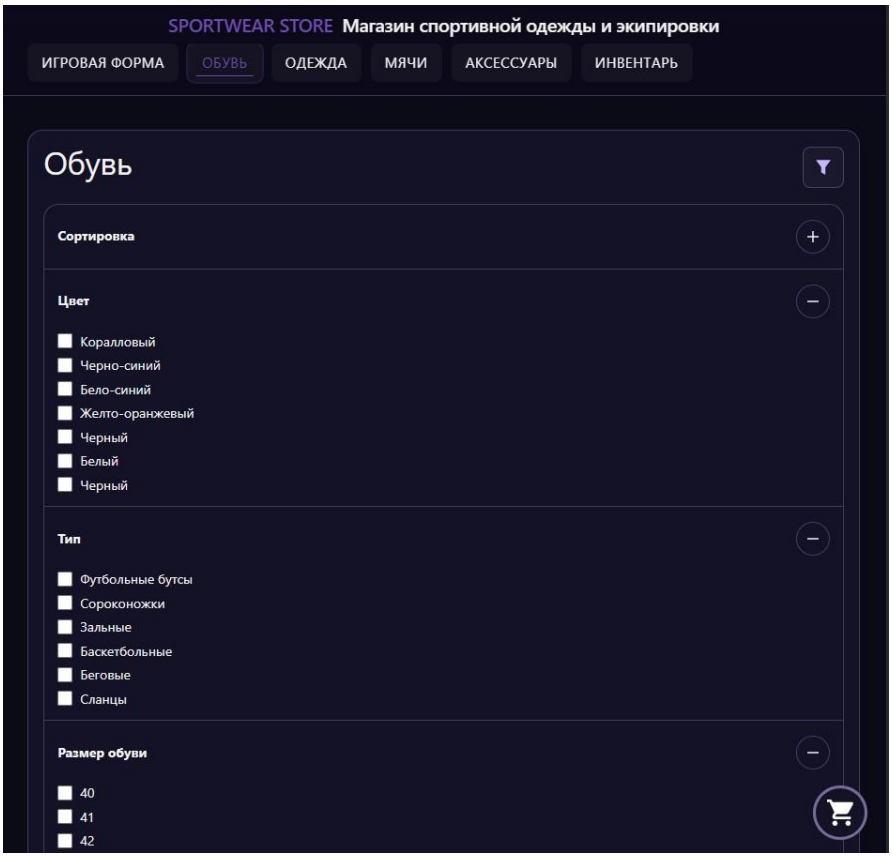


Рисунок 14 – Окно фильтрации

При нажатии на товар можно посмотреть подробную информацию о нем и добавить в корзину (рисунок 15).

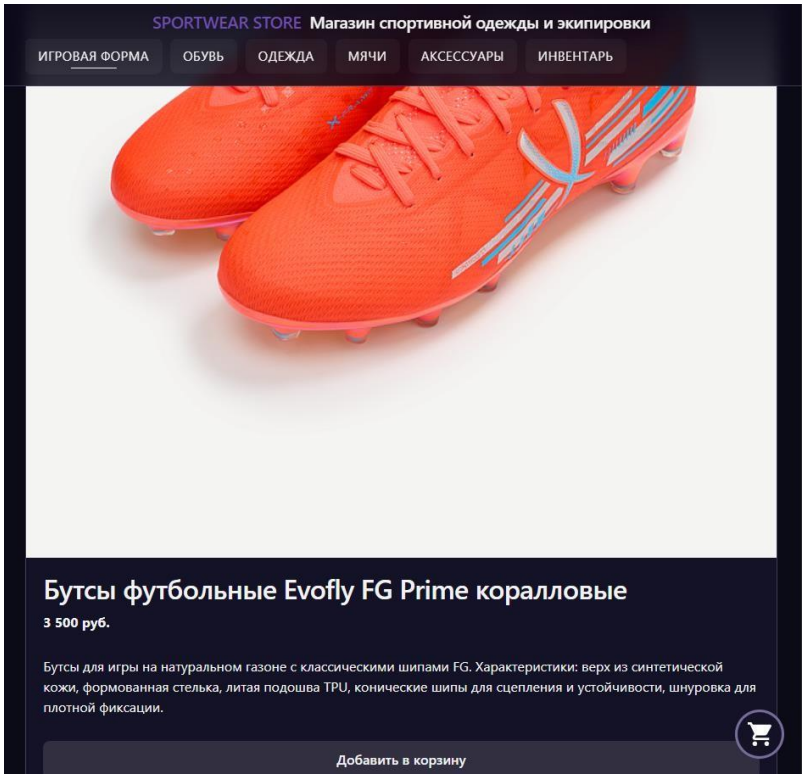


Рисунок 15 – Страница товара

После добавления товара в корзину, пользователь может оформить заказ, указав свой номер телефона (рисунок 16).

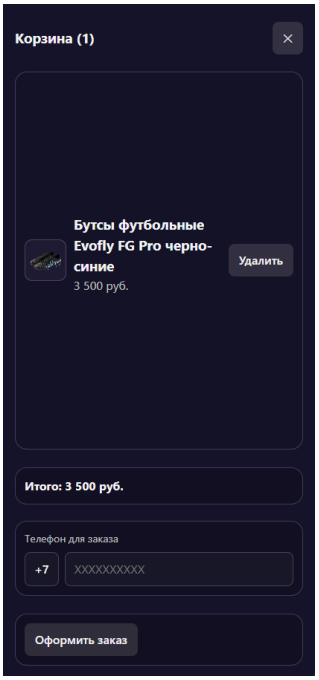


Рисунок 16 – Корзина

После оформления заказа появляется окно уведомления о покупке (рисунок 17).

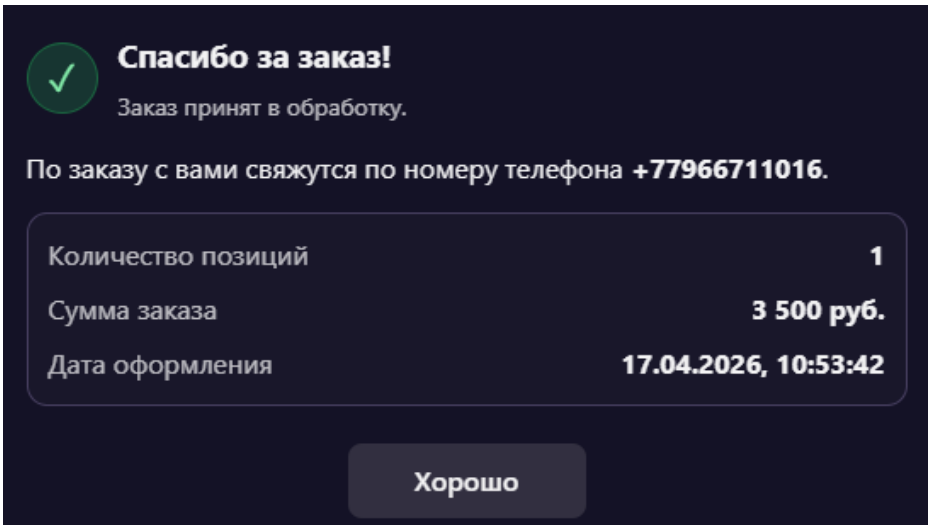


Рисунок 17 – Уведомление о покупке

1.4.4 Тестирование и отладка

В процессе работы с веб-приложением интернет-магазина пользователь может столкнуться с ситуациями, способными привести к некорректному отображению данных или нарушению логики оформления заказа. Подобные ситуации могут возникать как вследствие случайных ошибок ввода со стороны посетителя, так и из-за отсутствия необходимых данных на сервере. Для обеспечения стабильной работы и предсказуемого поведения интерфейса в приложении предусмотрен ряд механизмов обработки исключительных ситуаций, корректность функционирования которых была проверена путём составления и выполнения тестовых сценариев (тест-кейсов).

К числу наиболее вероятных ошибочных действий пользователя можно отнести следующие:

– попытка оформления заказа с пустой корзиной. Система должна блокировать отправку запроса на сервер и информировать посетителя о необходимости добавить хотя бы один товар;

– ввод некорректного номера телефона в форме оформления заказа. Пользователь может указать номер неполной длины, использовать буквенные

символы или специальные знаки. Клиентская часть приложения выполняет предварительную проверку формата ввода и не позволяет отправить заказ до тех пор, пока номер не будет соответствовать заданному шаблону (десять цифр, начинающихся с девятки);

– попытка отправить заказ при отсутствии соединения с сервером или при недоступности базы данных. В этом случае приложение должно корректно обработать возникшее исключение, вывести пользователю информативное сообщение о временной недоступности сервиса и не допустить потери данных о составе корзины;

– применение фильтров, не возвращающих ни одного товара. Если выбранная комбинация параметров (например, определённый цвет и размер) отсутствует в базе данных, система должна отобразить пустой каталог с соответствующим уведомлением, а не выдавать техническую ошибку;

– переход по несуществующему идентификатору товара или страницы каталога. В случае, если запрошенный ресурс не найден, приложение должно показать пользователю страницу с сообщением об ошибке и предложить вернуться в каталог.

Для проверки корректности работы приложения в перечисленных и других нештатных ситуациях были разработаны тест-кейсы, охватывающие как позитивные (ожидаемые), так и негативные сценарии взаимодействия пользователя с системой. Результаты тестирования подтвердили, что все предусмотренные обработчики исключений функционируют штатно, а интерфейс приложения предоставляет пользователю понятную обратную связь при возникновении ошибок.

2 Охрана труда

Разработка программного обеспечения, включая создание клиентской и серверной частей веб-приложения интернет-магазина спортивных товаров, относится к видам трудовой деятельности, связанным с интенсивным использованием персональных электронно-вычислительных машин (ПЭВМ). В связи с этим при организации рабочего места разработчика и планировании режима труда необходимо руководствоваться действующими нормативными документами, регламентирующими требования охраны труда при работе с вычислительной техникой.

Согласно установленным санитарно-эпидемиологическим правилам и нормативам, помещение, в котором осуществляется работа с персональным компьютером, должно соответствовать следующим требованиям:

– освещение рабочей зоны должно соответствовать установленным нормам. В случае отсутствия или недостаточности естественного освещения необходимо наличие расчетов, подтверждающих безопасность выполнения зрительных работ в условиях искусственного освещения;

– оконные проемы в помещениях, где эксплуатируются ПЭВМ, должны быть оборудованы регулируемыми светозащитными устройствами, такими как жалюзи или плотные занавеси, позволяющими предотвращать появление бликов на экранах мониторов;

– размещение компьютерной техники в подвальных помещениях, а также в местах, подверженных риску подтопления или воздействию повышенной влажности, не допускается;

– все рабочие места, оснащенные ПЭВМ, должны быть подключены к контуру защитного заземления в соответствии с требованиями электробезопасности;

– в помещениях с вычислительной техникой должна проводиться ежедневная влажная уборка, а также регулярное сквозное проветривание с периодичностью не реже одного раза в час;

– оптимальные параметры микроклимата на рабочем месте оператора ПЭВМ должны поддерживаться на уровне температуры воздуха +19°С — +21°С при относительной влажности 55–62%;

– конструкция рабочего стола должна обеспечивать возможность рационального размещения монитора, системного блока, периферийных устройств и документов. Допускается использование специализированной офисной мебели, соответствующей современным эргономическим требованиям.

При организации конкретных рабочих мест с персональными компьютерами необходимо соблюдать следующие правила их размещения:

– расстояние между тыльной поверхностью одного видеомонитора и экраном другого должно составлять не менее 2,0 метров. Расстояние между боковыми поверхностями соседних мониторов должно быть не менее 1,2 метра;

– размещать рабочие станции следует на расстоянии не менее 1,0 метра от приборов отопления и иных источников тепла. Прямое попадание солнечных лучей на экран монитора и поверхность рабочего стола не допускается;

– запрещается расположение рабочих мест в непосредственной близости от силовых кабелей электропитания, высоковольтных трансформаторов и технологического оборудования, способного создавать электромагнитные помехи и наводки, влияющие на стабильность работы вычислительной техники;

– при монтаже электропроводки и прокладке интерфейсных кабелей локальной сети следует избегать их прохождения по полу в зонах движения персонала. В случае технической необходимости прокладки кабелей по полу они должны быть убраны в защитные короба или кабель-каналы, исключающие механическое повреждение изоляции и риск спотыкания;

– расстояние от глаз пользователя до экрана монитора должно находиться в пределах 600–700 мм, но не должно быть менее 500 мм. Конкретное значение выбирается в зависимости от диагонали экрана и размера отображаемых символов;

– системный блок и монитор должны быть установлены таким образом, чтобы обеспечить беспрепятственный приток и отток воздуха через

вентиляционные отверстия. Расстояние от вентиляционных решеток корпуса до стены или иной преграды должно составлять не менее 10 сантиметров;

– искусственное освещение в помещениях, оборудованных ПЭВМ, должно быть выполнено в виде системы общего равномерного освещения. Уровень освещенности на поверхности рабочего стола в зоне размещения клавиатуры и документов должен составлять от 300 до 500 люкс. Осветительные приборы следует располагать таким образом, чтобы исключить появление световых бликов на полированной поверхности экрана. Яркость экрана монитора при этом не должна превышать 300 люкс.

В процессе разработки веб-приложения разработчик подвергается длительному статическому напряжению и зрительной нагрузке, что требует введения регламентированных перерывов для профилактики утомления. Для восьмичасовой рабочей смены перерывы устанавливаются в зависимости от категории выполняемых работ по уровню нервно-психического напряжения:

– для работников, выполняющих деятельность I категории (считывание информации с экрана), перерывы устанавливаются через 2 часа после начала смены и через 2 часа после обеденного перерыва. Продолжительность каждого перерыва составляет 15 минут;

– для работников II категории (ввод данных и диалог с системой) предусмотрен перерыв через 2 часа после начала работы и через 1,5–2,0 часа после обеденного перерыва. Продолжительность перерыва составляет 15 минут либо 10 минут через каждый час работы;

– для работников III категории (творческая работа в режиме диалога, напряженная умственная деятельность, к которой относится программирование) рекомендуется устанавливать перерыв через 1,5–2,0 часа после начала смены и через 1,5–2,0 часа после обеденного перерыва. Продолжительность перерыва должна составлять 20 минут либо 15 минут после каждого часа непрерывной работы.

В случае организации рабочего времени по графику 12-часовой смены регламентированные перерывы в первые 8 часов работы устанавливаются по

аналогии с восьмичасовым графиком в зависимости от категории работ. В течение последних четырех часов смены, независимо от категории и вида выполняемых работ, необходимо предусматривать перерыв продолжительностью не менее 15 минут через каждый час работы с ПЭВМ.

Заключение

В ходе выполнения данной выпускной квалификационной работы было спроектировано и разработано веб-приложение интернет-магазина спортивных товаров, предназначенное для автоматизации процессов представления ассортимента, фильтрации товаров и оформления заказов.

На начальном этапе был проведён анализ предметной области, в ходе которого выявлены основные проблемы, с которыми сталкиваются малые предприятия, специализирующиеся на розничной продаже спортивной экипировки: отсутствие собственного веб-сайта, использование устаревших или неадаптированных решений, высокая стоимость внедрения промышленных платформ электронной коммерции. Проведённый обзор существующих аналогов (Lamoda, Спортмастер, FreeRide) подтвердил наличие на рынке незаполненной ниши — отсутствие доступного, простого в развёртывании и ориентированного на нужды малого бизнеса веб-приложения.

На основании результатов анализа были сформулированы функциональные и технические требования к разрабатываемой системе, определены ключевые задачи и выбраны технологии для их реализации.

В итоге было принято решение разработать клиент-серверное веб-приложение со следующей архитектурой:

– клиентская часть реализована на языке JavaScript с использованием библиотеки React, что обеспечило создание динамичного, отзывчивого и удобного в сопровождении пользовательского интерфейса;

– серверная часть построена на языке Java с применением фреймворка Spring Boot, предоставляющего надёжную основу для реализации REST API и интеграции с базой данных;

– для хранения и управления данными о товарах, категориях, фильтрах и заказах применена СУБД PostgreSQL, а для автоматического управления структурой базы данных использован инструмент миграций Flyway;

– разработка как клиентской, так и серверной частей велась в единой среде Visual Studio Code, что позволило унифицировать рабочий процесс и ускорить выполнение поставленных задач.

На основе приведённых выше технологий удалось спроектировать, реализовать и протестировать веб-приложение со следующим функционалом:

– отображение многоуровневого каталога товаров с разбивкой на тематические категории (обувь, мячи, одежда, аксессуары, инвентарь);

– динамическая фильтрация товаров по набору параметров, специфичных для каждой категории (цвет, размер, тип, материал);

– сортировка товаров по цене и наименованию, а также постраничная навигация;

– просмотр детальной информации о товаре с его описанием и характеристиками;

– работа с корзиной покупок, позволяющая добавлять, просматривать и удалять выбранные позиции;

– оформление заказа с сохранением контактного номера телефона покупателя и автоматической фиксацией даты и суммы заказа;

– динамическое управление содержимым страниц и пунктами бокового меню через базу данных;

– отображение контактной информации магазина (телефон, email, адрес, социальные сети), загружаемой из базы данных.

Разработанное веб-приложение на текущий момент содержит весь необходимый базовый функционал для начала комфортной работы малого спортивного магазина в онлайн-формате. Применение свободно распространяемых технологий и фреймворков (Java, Spring Boot, React, PostgreSQL) позволило создать экономически доступное решение, не требующее приобретения дорогостоящих лицензий и способное функционировать на стандартном виртуальном хостинге.

Как и большинство программных продуктов, созданная система имеет потенциал для дальнейшего развития. В качестве перспективных направлений доработки можно выделить:

– добавление личного кабинета покупателя с возможностью просмотра истории заказов и отслеживания их статуса;

– интеграцию с платёжными шлюзами для приёма онлайн-оплаты;

– разработку административной панели с графическим интерфейсом для управления товарами, категориями и фильтрами без необходимости прямого доступа к базе данных;

– реализацию модуля аналитики для отслеживания наиболее востребованных товаров и формирования отчётов по продажам.

Созданное веб-приложение интернет-магазина спортивных товаров, призванное упростить взаимодействие покупателя с каталогом продукции и автоматизировать процесс приёма заказов, полностью готово к эксплуатации и может быть рекомендовано к внедрению в деятельность малых предприятий, работающих в сфере розничной продажи спортивной экипировки.

Список использованных источников

1. Хорстманн, Кей С. Java. Библиотека профессионала. Том 1. Основы. 11-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2022. – 928 с.

2. Рогов, Е. В. PostgreSQL 16 изнутри. – М.: ДМК Пресс, 2024. – 664 с.

3. Бэнкс, А., Порселло, Е. React и Redux: функциональная веб-разработка. – СПб.: Питер, 2021. – 336 с.

4. Уоллс, К. Spring Boot в действии. – М.: ДМК Пресс, 2020. – 346 с.

5. Официальная документация по языку программирования Java. [Электронный ресурс] / Режим доступа: https://docs.oracle.com/en/java/, свободный (дата обращения: 16.04.2026).

6. Официальная документация по библиотеке React. [Электронный ресурс] / Режим доступа: https://react.dev/reference/react, свободный (дата обращения: 16.04.2026).

7. Официальная документация по фреймворку Spring Boot. [Электронный ресурс] / Режим доступа: https://docs.spring.io/spring-boot/documentation.html, свободный (дата обращения: 16.04.2026).

8. Официальная документация по СУБД PostgreSQL. [Электронный ресурс] / Режим доступа: https://www.postgresql.org/docs/, свободный (дата обращения: 16.04.2026).

9. Официальная документация по инструменту сборки Vite. [Электронный ресурс] / Режим доступа: https://vitejs.dev/guide/, свободный (дата обращения: 16.04.2026).

10. Официальная документация по инструменту миграций Flyway. [Электронный ресурс] / Режим доступа: https://documentation.red-gate.com/flyway, свободный (дата обращения: 16.04.2026).

11. Официальная документация по среде разработки Visual Studio Code. [Электронный ресурс] / Режим доступа: https://code.visualstudio.com/docs, свободный (дата обращения: 16.04.2026)

Приложение А

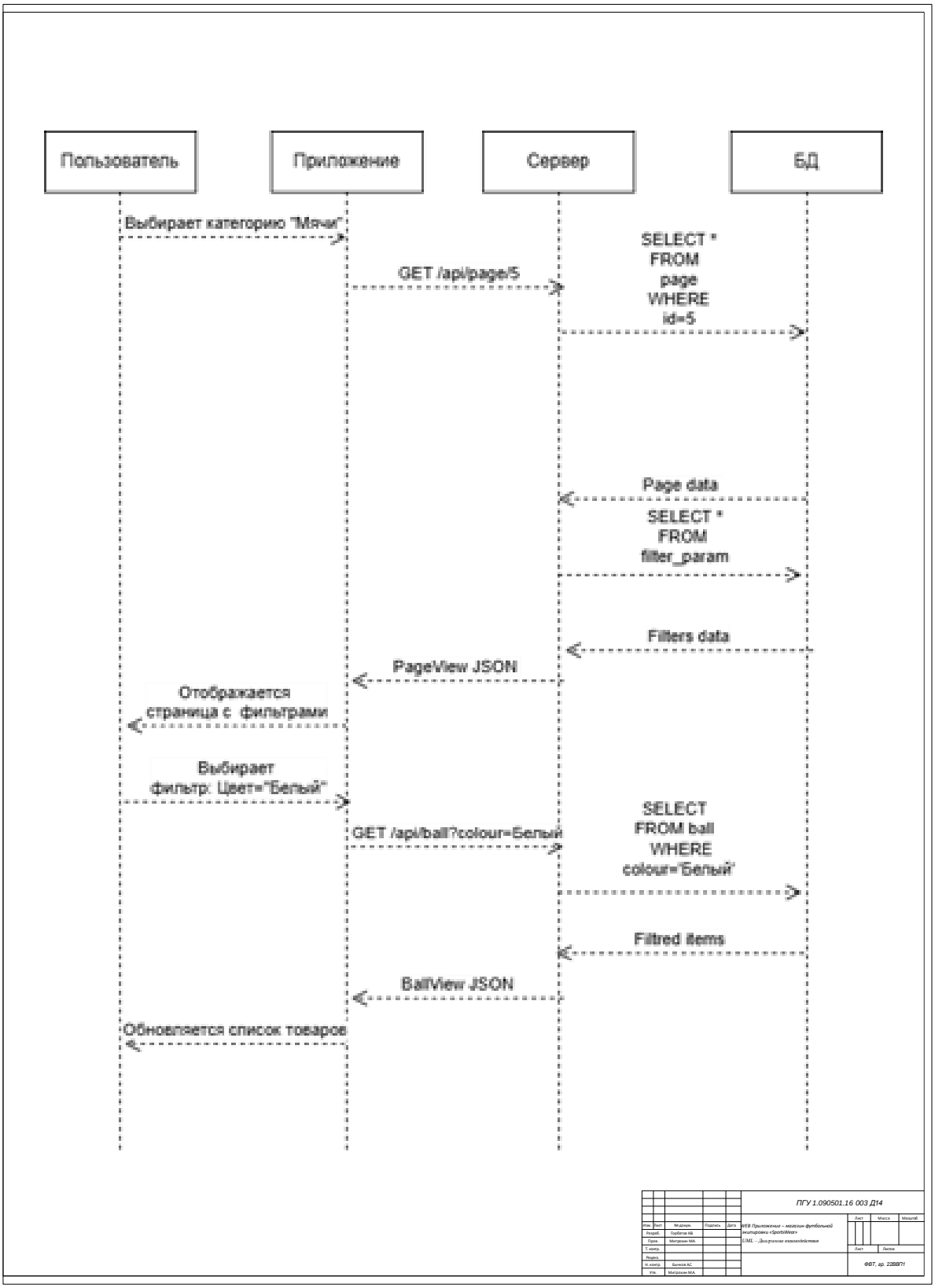
(обязательное)

WEB приложение – магазин футбольной экипировки «Sportwear Store»

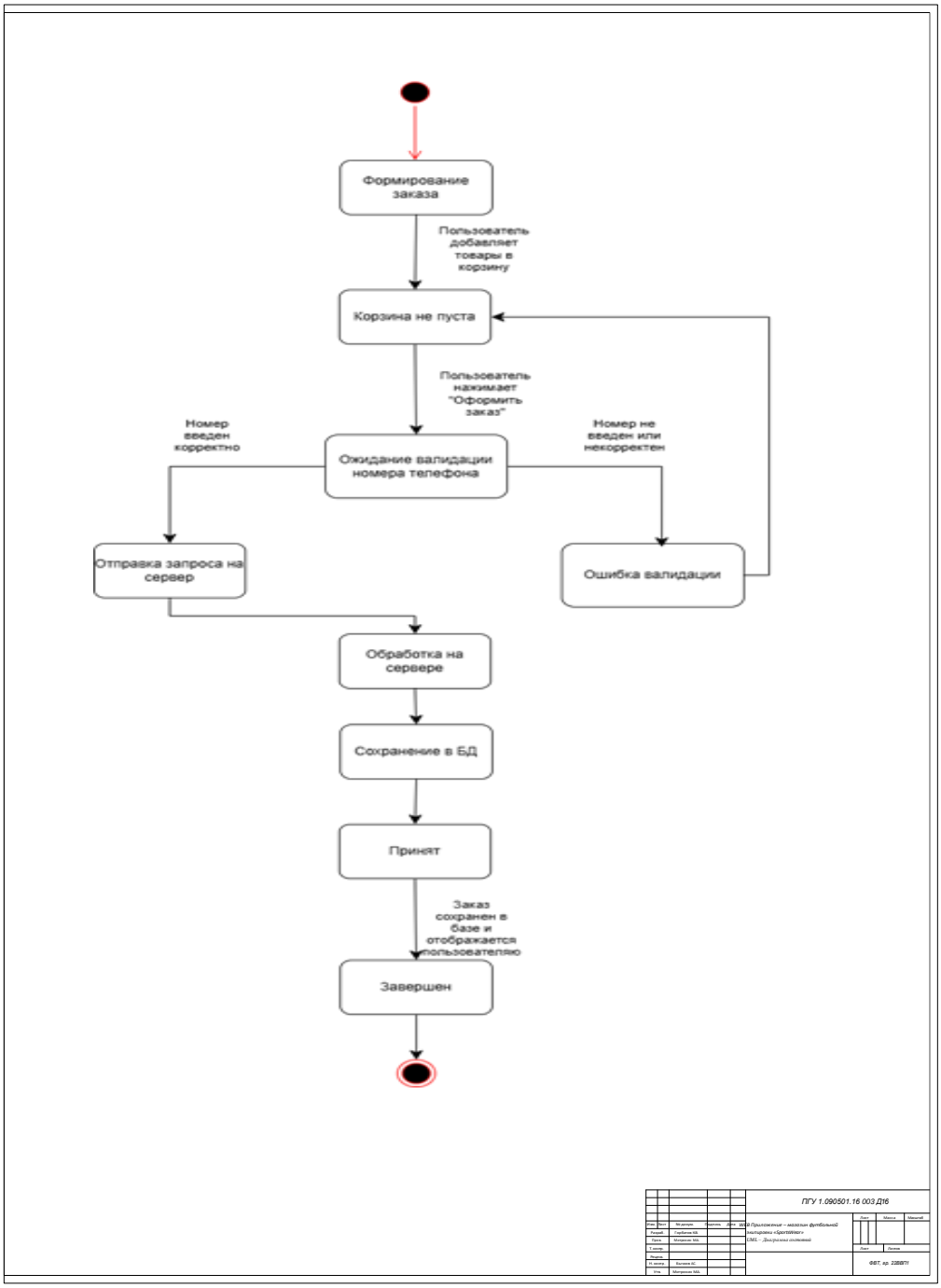
UML-диаграммы

UML – диаграмма вариантов использования

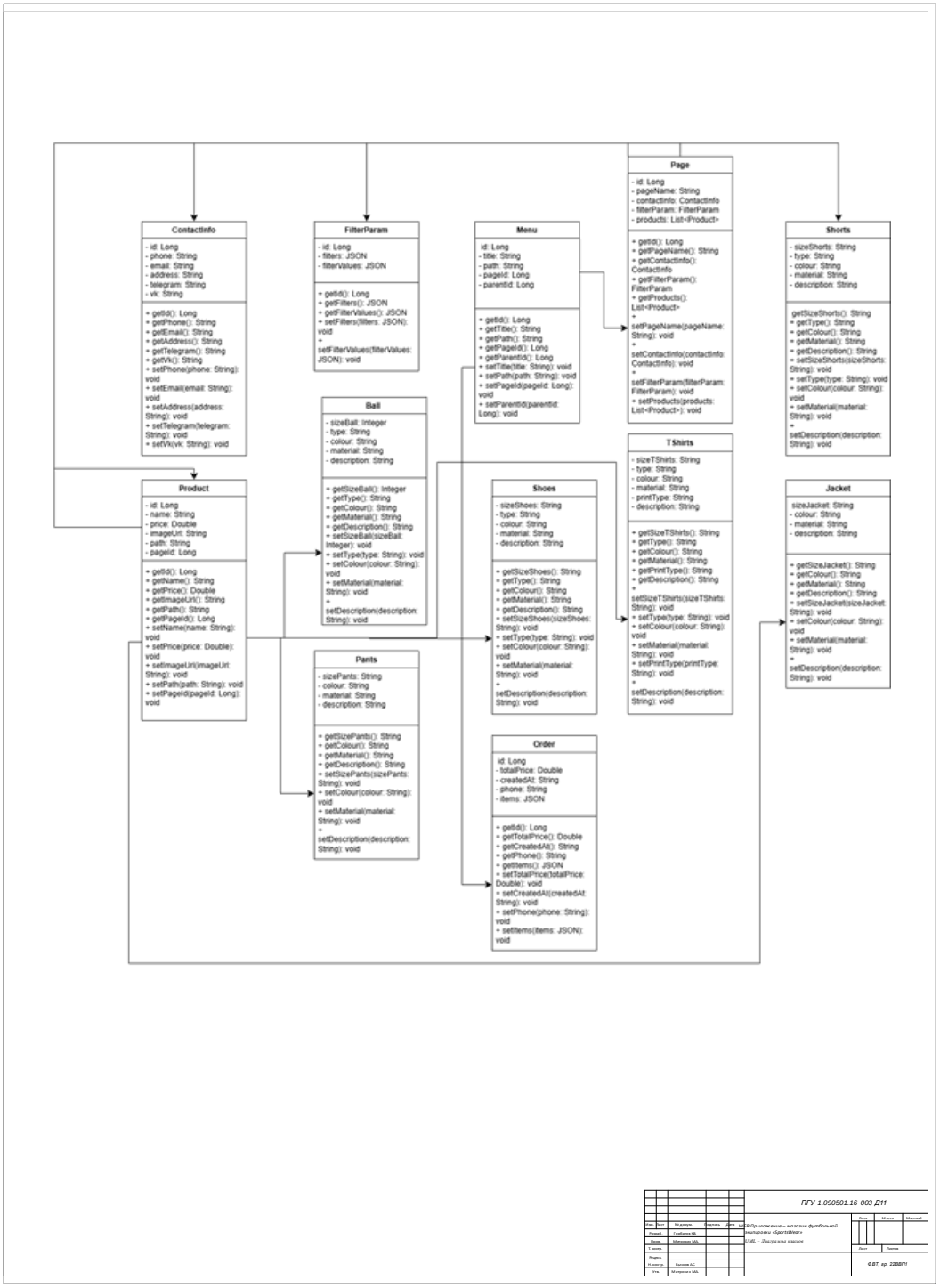
UML – диаграмма взаимодействия



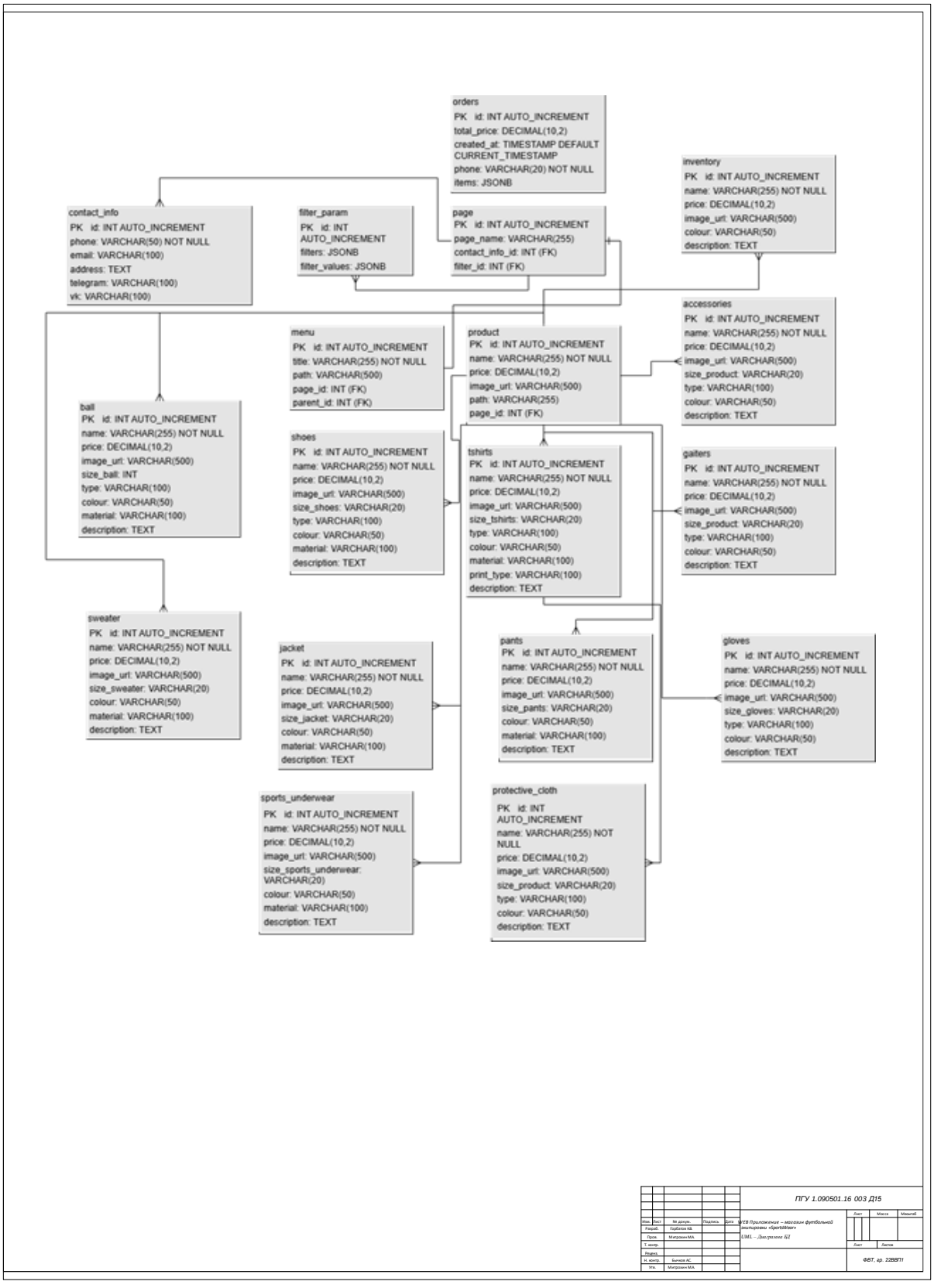
UML – диаграмма состояний



UML – диаграмма классов



UML – диаграмма БД



ПГУ 1.090501.16 003 Д15

ИЕВ Приложение – магазин футбольной экипировки «SportMaster»

UML – Диаграмма БД

ФВТ, гр. 23ВВП1

Приложение Б

(обязательное)

WEB приложение – магазин футбольной экипировки «Sportwear Store»

Листинги программы

Файл PageItems.jsx:

```
import { useCallback, useEffect, useMemo, useState } from "react";
import { Link, useLocation, useParams } from "react-router-dom";
import {
  getAccessories,
  getBalls,
  getGaiters,
  getGloves,
  getInventory,
  getJackets,
  getPageById,
  getPants,
  getProtectiveCloth,
  getShoes,
  getShorts,
  getSportsUnderwear,
  getSweaters,
  getTShirts,
} from "../api/shopApi";
import { useCart } from "../context/CartContext";
import { resolveImageUrl } from "../shared/resolveImageUrl";

const pageRequestById = {
  3: getShoes,
  5: getBalls,
  6: getAccessories,
  7: getInventory,
  8: getTShirts,
  9: getProtectiveCloth,
  10: getGloves,
  11: getGaiters,
  12: getShorts,
  13: getJackets,
  14: getPants,
  15: getSweaters,
  16: getSportsUnderwear,
};

const pageRequestByName = {
  обувь: getShoes,
  мячи: getBalls,
  инвентарь: getInventory,
  щитки: getProtectiveCloth,
  "вратарские перчатки": getGloves,
  "гетры и гольфы": getGaiters,
  футболки: getTShirts,
  шорты: getShorts,
  куртки: getJackets,
  ветровки: getJackets,
  брюки: getPants,
  джемперы: getSweaters,
```

```
  олимпийки:  getSweaters,
  толстовки:  getSweaters,
  худи: getSweaters,
  "спортивное бельё": getSportsUnderwear,
  "спортивное белье": getSportsUnderwear,
  защита: getProtectiveCloth,
  перчатки: getGloves,
  гетры: getGaiters,
  аксессуары: getAccessories,
};

const normalizers = {
  default: (item) => ({
    id: item.id,
    title: item.name,
    price: item.price,
    image: resolveImageUrl(item.imageUrl),
    path: item.path,
    redirectPageId:  item.redirectPageId,
  }),
};

const normalizeByPage = (pageId) => normalizers[pageId] ||
normalizers.default;

const resolveRequestByPageName = (pageName) => {
  const normalizedName = String(pageName || "").toLowerCase();
  const  matched  =  Object.entries(pageRequestByName).find(([keyword])
=>
    normalizedName.includes(keyword),
  );
  return matched?.[1];
};

const toPageLink = (item) => {
  const rawPath = String(item?.path || "");
  const match = /^\/api\/page\/(\d+)$/.exec(rawPath);
  if (match) return `/page/${match[1]}`;
  if (typeof item?.redirectPageId === "number")
    return `/page/${item.redirectPageId}`;
  if (typeof item?.id === "number") return `/page/${item.id}`;
  return "/";
};

const productCategoryByPageId = {
  3: "shoes",
  5: "balls",
  6: "accessories",
  7: "inventory",
  8: "tshirts",
  9: "protective_cloth",
  10: "gloves",
  11: "gaiters",
```

```
  12: "shorts",
  13: "jackets",
  14: "pants",
  15: "sweaters",
  16: "sports_underwear",
};

function PageItems() {
  const PAGE_SIZE = 12;
  const { id } = useParams();
  const location = useLocation();
  const { addItem } = useCart();
  const pageId = Number(id);

  const [items, setItems] = useState([]);
  const [totalCount, setTotalCount] = useState(0);
  const [title, setTitle] = useState(`Страница ${id}`);
  const [description, setDescription] = useState("");
  const [pageFilters, setPageFilters] = useState([]);
  const [filterValues, setFilterValues] = useState({});
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");
  const [filters, setFilters] = useState({ sort: "" });
  const [expandedFilters, setExpandedFilters] = useState({});
  const [requestFn, setRequestFn] = useState(() =>
pageRequestById[pageId]);
  const [requestResolvedForPageId, setRequestResolvedForPageId] =
useState(null);
  const [pageProducts, setPageProducts] = useState([]);
  const [filtersOpen, setFiltersOpen] = useState(false);
  const [currentPage, setCurrentPage] = useState(1);

  const toFilterLabel = (filterKey) => {
    const labels = {
      color: "Цвет",
      colour: "Цвет",
      type: "Тип",
      material: "Материал",
      sizeShoes: "Размер обуви",
      sizeBall: "Размер мяча",
      sizeTShirts: "Размер футболки",
      sizeShorts: "Размер шорт",
      sizeJacket: "Размер куртки",
      sizePants: "Размер брюк",
      sizeSweater: "Размер",
      sizeSportsUnderwear: "Размер белья",
      sizeProduct: "Размер изделия",
      sizeGloves: "Размер перчаток",
    };
    return labels[filterKey] || filterKey;
  };

  const buildParamsFromFilters = useCallback((state) => {
```

```
    const sortMapping = {
      price_asc: { sortBy: "price", direction: "asc" },
      price_desc: { sortBy: "price", direction: "desc" },
      name_asc: { sortBy: "name", direction: "asc" },
      name_desc: { sortBy: "name", direction: "desc" },
    };

    const baseParams = Object.fromEntries(
      pageFilters
        .map((key) => [key, Array.isArray(state[key]) ? state[key] :
[]])
        .filter(([, values]) => values.length > 0)
        .map(([key, values]) => [key, values.join(",")]),
    );

    return {
      ...baseParams,
      ...(sortMapping[state.sort] || {}),
    };
  }, [pageFilters]);

  const parseListResponse = useCallback((data) => {
    if (Array.isArray(data)) {
      return { items: data, count: data.length };
    }
    if (data && typeof data === "object") {
      const list = Array.isArray(data.items) ? data.items : [];
      const count = Number(data.count);
      return {
        items: list,
        count: Number.isNaN(count) ? list.length : count,
      };
    }
    return { items: [], count: 0 };
  }, []);

  useEffect(() => {
    async function loadPageMeta() {
      try {
        const page = await getPageById(pageId);
        if (page?.pageName) setTitle(page.pageName);
        if (!page?.pageName && page?.title) setTitle(page.title);
        if (page?.description) setDescription(page.description);
        const resolvedRequest =
          pageRequestById[pageId] ||
resolveRequestByPageName(page?.pageName);
        setRequestFn(() => resolvedRequest);
        setRequestResolvedForPageId(resolvedRequest ? pageId :
null);
        const isPageProducts =
          Array.isArray(page?.products) && page.products.length > 0;
        setPageProducts(isPageProducts ? page.products : []);
```

```
        const nextPageFilters =
Array.isArray(page?.filterParam?.filters)
          ? page.filterParam.filters
          : [];
        const nextFilterValues =
          page?.filterParam?.filterValues &&
          typeof page.filterParam.filterValues === "object"
            ? page.filterParam.filterValues
            : {};
        setPageFilters(nextPageFilters);
        setFilterValues(nextFilterValues);
        setFilters((prev) => {
          const next = { sort: prev.sort || "" };
          nextPageFilters.forEach((key) => {
            next[key] = Array.isArray(prev[key]) ? prev[key] : [];
          });
          return next;
        });
        setExpandedFilters(
          Object.fromEntries(nextPageFilters.map((key)  =>  [key,
false])),
        );
        if (!resolvedRequest && isPageProducts) {
          setItems(page.products);
          setTotalCount(page.products.length);
          setLoading(false);
          setError("");
        } else if (!resolvedRequest) {
          setItems([]);
          setTotalCount(0);
          setLoading(false);
          setError(
            `Для page id ${pageId} не найден endpoint списка в
Swagger.`,
          );
        }
      } catch (requestError) {
        setRequestResolvedForPageId(null);
        setError(requestError.message || "Не удалось загрузить
страницу.");
      }
    }

    loadPageMeta();
  }, [pageId]);

  const loadItems = useCallback(async (params, requestOverride) => {
    const effectiveRequest = requestOverride || requestFn;
    try {
      setLoading(true);
      setError("");
      if (!effectiveRequest) {
        setItems([]);
```

```
        setError(`Для page id ${pageId} не найден endpoint списка в
Swagger.`);
        return;
      }
      const queryParams = {
        size: PAGE_SIZE,
        page: Math.max(0, currentPage - 1),
        ...(params || {}),
      };
      const data = await effectiveRequest(queryParams);
      const parsed = parseListResponse(data);
      setItems(parsed.items);
      setTotalCount(parsed.count);
    } catch (requestError) {
      setError(requestError.message || "Failed to load category
items.");
      setItems([]);
      setTotalCount(0);
    } finally {
      setLoading(false);
    }
  }, [PAGE_SIZE, currentPage, parseListResponse, requestFn,
pageId]);

  useEffect(() => {
    setPageFilters([]);
    setFilterValues({});
    setFilters({ sort: "" });
    setExpandedFilters({});
    setRequestFn(undefined);
    setRequestResolvedForPageId(null);
    setPageProducts([]);
    setFiltersOpen(false);
    setCurrentPage(1);
    setTotalCount(0);
  }, [pageId]);

  const onSortChange = (event) => {
    const sort = event.target.value;
    setFilters((prev) => {
      setCurrentPage(1);
      return { ...prev, sort };
    });
  };

  const onCheckboxChange = (field, option) => (event) => {
    const checked = event.target.checked;
    setFilters((prev) => {
      const current = Array.isArray(prev[field]) ? prev[field] : [];
      const nextValues = checked
        ? [...current, option]
        : current.filter((value) => value !== option);
      setCurrentPage(1);
```

```
      return { ...prev, [field]: nextValues };
    });
  };

  const toggleFilterGroup = (field) => {
    setExpandedFilters((prev) => ({ ...prev, [field]: !prev[field]
}));
  };

  const onResetFilters = () => {
    const empty = {
      sort: "",
      ...Object.fromEntries(pageFilters.map((key) => [key, []])),
    };
    setFilters(empty);
    setCurrentPage(1);
  };

  const normalizedItems = useMemo(() => {
    const normalizeItem = normalizeByPage(pageId);
    return items.map(normalizeItem);
  }, [items, pageId]);
  const totalPages = Math.max(1, Math.ceil(totalCount / PAGE_SIZE));
  const safePage = Math.min(currentPage, totalPages);
  const paginatedItems = normalizedItems;
  const paginationPages = Array.from(
    new Set([1, safePage - 1, safePage, safePage + 1, totalPages]),
  ).filter((page) => page >= 1 && page <= totalPages).sort((a, b) =>
a - b);

  useEffect(() => {
    if (currentPage !== safePage) setCurrentPage(safePage);
  }, [currentPage, safePage]);

  useEffect(() => {
    if (!requestFn || requestResolvedForPageId !== pageId) return;
    loadItems(buildParamsFromFilters(filters));
  }, [requestFn, filters, currentPage, buildParamsFromFilters,
loadItems, requestResolvedForPageId, pageId]);

  const productCategoryKey = productCategoryByPageId[pageId] || "";

  return (
    <section className="panel">
      <div  className="title-row">
        <h1 className="title">{title}</h1>
        {requestFn && (
          <button
            className="filters-toggle"
            type="button"
            onClick={() => setFiltersOpen((prev) => !prev)}
            aria-label={filtersOpen ? "Скрыть фильтры и сортировку"
: "Показать фильтры и сортировку"}
```

```
            title={filtersOpen ? "Скрыть фильтры и сортировку" :
"Показать фильтры и сортировку"}
          >
            <svg
              className="filters-toggle-icon"
              viewBox="0 0 16 16"
              xmlns="http://www.w3.org/2000/svg"
              aria-hidden="true"
            >
              <path
                fillRule="evenodd"
                clipRule="evenodd"
                d="M9.969 8.006v7.038L6.03 12.917v-4.91L1.084
1.043h14.003L9.97 8.006z"
                fill="currentColor"
              />
            </svg>
          </button>
        )}
      </div>
      {description && <p className="muted">{description}</p>}

      <div
        className={`catalog-layout ${!requestFn || (requestFn &&
!filtersOpen) ? "collapsed" : ""}`}
      >
        <section  className="catalog-content">
          {loading && <div className="state">Loading items...</div>}
          {error && <div className="state error">{error}</div>}

          {!loading && !error && pageProducts.length > 0 &&
!requestFn && (
            <ul  className="category-tiles">
              {normalizedItems.map((item)  =>  (
                <li key={String(item.id)}>
                  <Link className="category-tile-link"
to={toPageLink(item)}>
                    {item.image && (
                      <img
                        className="category-tile-image"
                        src={item.image}
                        alt={item.title || "Category image"}
                      />
                    )}
                    <span  className="category-tile-title">
                      {item.title || "Категория"}
                    </span>
                  </Link>
                </li>
              ))}
            </ul>
          )}
```

```
          {!loading && !error && !(pageProducts.length > 0 &&
!requestFn) && (
            <>
              <ul  className="items-list">
                {paginatedItems.map((item)  =>  (
                <li className="item" key={String(item.id)}>
                  <Link
                    className="item-link"
                    to={`/product/${item.id}?category=${productCateg
oryKey}`}
                    state={{ from:
`${location.pathname}${location.search}`  }}
                  >
                    {item.image && (
                      <img
                        className="item-image"
                        src={item.image}
                        alt={item.title || "Product image"}
                      />
                    )}
                    <div  className="item-body">
                      <h3  className="item-title">
                        {item.title || "Untitled item"}
                      </h3>
                      <p  className="item-price">
                        {typeof item.price === "number"
                          ?  `${item.price.toLocaleString("ru-RU")}
₽`
                          : "Price not specified"}
                      </p>
                    </div>
                  </Link>
                  <div  className="item-actions">
                    <button
                      type="button"
                      className="add-btn"
                      onClick={() => addItem(item)}
                    >
                      В корзину
                    </button>
                  </div>
                </li>
                ))}
              </ul>
              {totalPages > 1 && (
                <nav  className="pagination"  aria-label="Пагинация
каталога">
                  {paginationPages.map((page, index) => {
                    const prevPage = paginationPages[index - 1];
                    const shouldShowGap = index > 0 && page -
prevPage > 1;

                    return (
```

```
                        <div key={`page-slot-${page}`}
className="pagination-slot">
                          {shouldShowGap && <span
className="pagination-ellipsis">…</span>}
                          <button
                            type="button"
                            className={`pagination-btn ${page ===
safePage ? "active" : ""}`}
                            onClick={() => setCurrentPage(page)}
                          >
                            {page}
                          </button>
                        </div>
                      );
                    })}
                  </nav>
                )}
              </>
            )}
          </section>

          {requestFn && filtersOpen && (
            <aside  className="filters-sidebar">
              <form  className="filters-form">
                <div  className="filters-group">
                  <button
                    type="button"
                    className="filters-group-toggle"
                    onClick={() => toggleFilterGroup("sort")}
                  >
                    Сортировка
                  </button>
                  {expandedFilters.sort && (
                    <div  className="filters-options">
                      <label  className="filters-option">
                        <input
                          type="radio"
                          name="sort"
                          value=""
                          checked={filters.sort === ""}
                          onChange={onSortChange}
                        />
                        <span>По умолчанию</span>
                      </label>
                      <label  className="filters-option">
                        <input
                          type="radio"
                          name="sort"
                          value="price_asc"
                          checked={filters.sort === "price_asc"}
                          onChange={onSortChange}
                        />
                        <span>Цена: по возрастанию</span>
```

```
                    </label>
                    <label  className="filters-option">
                      <input
                        type="radio"
                        name="sort"
                        value="price_desc"
                        checked={filters.sort === "price_desc"}
                        onChange={onSortChange}
                      />
                      <span>Цена: по убыванию</span>
                    </label>
                    <label  className="filters-option">
                      <input
                        type="radio"
                        name="sort"
                        value="name_asc"
                        checked={filters.sort === "name_asc"}
                        onChange={onSortChange}
                      />
                      <span>Название:  А-Я</span>
                    </label>
                    <label  className="filters-option">
                      <input
                        type="radio"
                        name="sort"
                        value="name_desc"
                        checked={filters.sort === "name_desc"}
                        onChange={onSortChange}
                      />
                      <span>Название:  Я-А</span>
                    </label>
                  </div>
                )}
              </div>
              {pageFilters.map((key) => (
                <div  className="filters-group"  key={key}>
                  <button
                    type="button"
                    className="filters-group-toggle"
                    onClick={() => toggleFilterGroup(key)}
                  >
                    {toFilterLabel(key)}
                  </button>
                  {expandedFilters[key] && (
                    <div  className="filters-options">
                      {(Array.isArray(filterValues[key]) ?
filterValues[key] : [])
                        .map((option) => {
                          const optionValue = option;
                          return (
                          <label key={optionValue}
className="filters-option">
                            <input
```

```
                                type="checkbox"
                                checked={(filters[key] ||
[]).includes(optionValue)}
                                onChange={onCheckboxChange(key,
optionValue)}
                              />
                              <span>{optionValue}</span>
                            </label>
                            );
                          })}
                        </div>
                      )}
                    </div>
                  ))}
                  <button type="button" onClick={onResetFilters}>
                    Сбросить
                  </button>
                </form>
              </aside>
            )}
          </div>
        </section>
    );
}

export default PageItems;
```

Файл ProductPage.jsx:

```
import { useEffect, useState } from 'react'
import { Link, useLocation, useParams, useSearchParams } from
'react-router-dom'
import { getProductById } from '../api/shopApi'
import { useCart } from '../context/CartContext'
import LoadingState from '../components/LoadingState'
import ErrorState from '../components/ErrorState'
import { resolveImageUrl } from '../shared/resolveImageUrl'

function ProductPage() {
  const { id } = useParams()
  const location = useLocation()
  const [searchParams] = useSearchParams()
  const categoryKey = searchParams.get('category') || ''
  const { addItem } = useCart()
  const [product, setProduct] = useState(null)
  const [loading, setLoading] = useState(true)
  const [error, setError] = useState('')

  useEffect(() => {
    async function loadProduct() {
```

```
      try {
        setLoading(true)
        setError('')
        const data = await getProductById(id, categoryKey)
        setProduct(data)
      } catch (requestError) {
        setError(requestError.message || 'Не удалось загрузить
товар')
      } finally {
        setLoading(false)
      }
    }

    loadProduct()
  }, [id, categoryKey])

  if (loading) return <LoadingState text="Загрузка карточки
товара..." />
  if (error) return <ErrorState text={error} />
  if (!product) return <ErrorState text="Товар не найден" />

  const characteristics = product.characteristics || {}
  const pageByCategory = {
    shoes: 3,
    balls: 5,
    ball: 5,
    accessories: 6,
    inventory: 7,
    tshirts: 8,
    protective_cloth:  9,
    gloves: 10,
    gaiters: 11,
    shorts: 12,
    jackets: 13,
    pants: 14,
    sweaters: 15,
    sports_underwear:  16,
  }
  const fallbackPageId = pageByCategory[categoryKey] || 1
  const backTo = location.state?.from || `/page/${fallbackPageId}`
  const productImageUrl = resolveImageUrl(product.imageUrl) ||
'https://placehold.co/640x400?text=SportWear'

  return (
    <section>
      <Link  to={backTo}  className="back-link">
        ← Назад в каталог
      </Link>

      <div  className="product-page">
        <img
          src={productImageUrl}
          alt={product.name}
```

```
          className="product-page-image"
          onError={(event) => {
            event.currentTarget.src =
'https://placehold.co/640x400?text=SportWear'
          }}
        />

        <div  className="product-page-content">
          <h2>{product.name}</h2>
          <p
className="price">{Number(product.price).toLocaleString('ru-RU')}
руб.</p>
          <p>{product.description || 'Описание товара пока не
заполнено.'}</p>

          {/*
          <div className="specs">
            <h3>Характеристики</h3>
            <ul>
              <li>Вид спорта: {product.sport || '—'}</li>
              <li>Категория: {product.category || '—'}</li>
              <li>Тип: {product.type || '—'}</li>
              <li>Цвет: {product.colour || '—'}</li>
              <li>Материал: {product.material || '—'}</li>
              <li>Размер: {product.sizeBall || product.sizeShoes ||
'—'}</li>
              <li>Бренд: {characteristics.brand || '—'}</li>
              <li>Сезон: {characteristics.season || '—'}</li>
              <li>Посадка: {characteristics.fit || '—'}</li>
              <li>Уход: {characteristics.care || '—'}</li>
            </ul>
          </div>
          */}

          <button className="add-btn" onClick={() =>
addItem(product)}>
            Добавить в корзину
          </button>
        </div>
      </div>
    </section>
  )
}

export default ProductPage
```

Файл App.jsx:

```
import { useEffect, useState } from "react";
import {
```

```
  Link,
  NavLink,
  Navigate,
  Route,
  Routes,
  useLocation,
} from "react-router-dom";
import "./App.css";
import PageItems from "./pages/PageItems";
import ProductPage from "./pages/ProductPage";
import { getMenuCategories } from "./api/shopApi";
import CartDrawer from "./components/CartDrawer";
import { useCart } from "./context/CartContext";

function App() {
  const location = useLocation();
  const { totalItems } = useCart();
  const contactInfo = {
    phone: "+7 999 123-45-67",
    email: "info@example.com",
    address: "Пенза, ул. Красная 1",
    telegram: "@example_telegram",
    vk: "vk.com/example",
  };
  const [menu, setMenu] = useState([]);
  const [menuLoading, setMenuLoading] = useState(true);
  const [menuError, setMenuError] = useState("");
  const [cartOpen, setCartOpen] = useState(false);
  const [routeLoading, setRouteLoading] = useState(true);
  const homeLink = "/page/1";

  useEffect(() => {
    async function loadMenu() {
      try {
        setMenuLoading(true);
        setMenuError("");
        const data = await getMenuCategories();
        setMenu(Array.isArray(data) ? data : []);
      } catch (requestError) {
        setMenu([]);
        setMenuError(requestError.message || "Не удалось загрузить
меню.");
      } finally {
        setMenuLoading(false);
      }
    }

    loadMenu();
  }, []);

  useEffect(() => {
    setRouteLoading(true);
    const timerId = window.setTimeout(() => {
```

```
      setRouteLoading(false);
    }, 280);

    return () => {
      window.clearTimeout(timerId);
    };
  }, [location.pathname, location.search]);

  return (
    <div className="app">
      <header  className="site-header">
        <div  className="container  header-top">
          <div  className="brand-row">
            <Link className="brand-kicker" to={homeLink}>
              SPORTWEAR STORE
            </Link>
            <h1  className="brand-title">
              Магазин спортивной одежды и экипировки
            </h1>
          </div>
          {!menuLoading && !menuError && menu.length > 0 && (
            <nav  className="header-nav">
              {menu
                .filter(
                  (item) =>
                    String(item.title || "").toLowerCase() !==
"главная",
                )
                .map((item) => (
                  <NavLink
                    key={item.id || item.pageId}
                    to={`/page/${item.pageId || item.id}`}
                    className={({ isActive }) =>
                      `header-link ${isActive ? "active" : ""}`
                    }
                  >
                    {item.title}
                  </NavLink>
                ))}
            </nav>
          )}
        </div>
      </header>

      <main  className="container  page-content">
        <Routes>
          <Route path="/" element={<Navigate to="/page/1" replace
/>} />
          <Route path="/page/:id" element={<PageItems />} />
          <Route path="/product/:id" element={<ProductPage />} />
        </Routes>
      </main>
```

```
      {routeLoading && (
        <div className="page-loading-overlay" role="status" aria-
live="polite">
          <div  className="page-loading-spinner"  aria-hidden="true"
/>
          <span>Загрузка  страницы...</span>
        </div>
      )}

      <footer  className="site-footer">
        <div  className="container  footer-grid">
          <section  className="footer-block">
            <h3>Контакты</h3>
            <p>
              Телефон:{" "}
              <a href={`tel:${contactInfo.phone.replace(/\s+/g,
"")}`}>
                {contactInfo.phone}
              </a>
            </p>
            <p>
              Email:{" "}
              <a
href={`mailto:${contactInfo.email}`}>{contactInfo.email}</a>
            </p>
            <p>Адрес: {contactInfo.address}</p>
          </section>

          <section  className="footer-block">
            <h3>Мы в соцсетях</h3>
            <p>
              Telegram:{" "}
              <a
href={`https://t.me/${contactInfo.telegram.replace("@",  "")}`}>
                {contactInfo.telegram}
              </a>
            </p>
            <p>
              VK: <a
href={`https://${contactInfo.vk}`}>{contactInfo.vk}</a>
            </p>
            <p className="footer-muted">
              Поддержка: ежедневно с 09:00 до 21:00
            </p>
          </section>

          <section  className="footer-block">
            <h3>Покупателям</h3>
            <p>Доставка по Пензе и области.</p>
            <p>Самовывоз из пункта выдачи.</p>
            <p  className="footer-muted">
              Ответим по заказу в течение 15 минут в рабочее время.
            </p>
```

```
          </section>
        </div>
        <div  className="container  footer-bottom">
          <small>SportWear Store, {new
Date().getFullYear()}.</small>
        </div>
      </footer>
      {!cartOpen && (
        <button
          type="button"
          className={`cart-fab ${totalItems > 0 ? "has-items" : "is-
empty"}`}
          onClick={() => setCartOpen(true)}
          aria-label={
            totalItems > 0
              ? `Открыть корзину, товаров: ${totalItems}`
              : "Открыть пустую корзину"
          }
          title={
            totalItems > 0
              ? `Товаров в корзине: ${totalItems}`
              : "Корзина пустая"
          }
        >
          <img
            src="/cart-icon.svg"
            alt=""
            aria-hidden="true"
            className="cart-fab-image"
          />
          {totalItems > 0 && (
            <span  className="cart-fab-badge">{totalItems}</span>
          )}
        </button>
      )}
      <CartDrawer open={cartOpen} onClose={() => setCartOpen(false)}
/>
    </div>
  );
}

export default App;

Файл «CartContext.jsx»
import { createContext, useContext, useEffect, useMemo, useState }
from 'react'
import { resolveImageUrl } from '../shared/resolveImageUrl'

const CART_STORAGE_KEY = 'sportwear-cart-v1'
const CartContext = createContext(null)

function makeCartItemId() {
  if (typeof crypto !== 'undefined' && crypto.randomUUID) {
```

```
    return crypto.randomUUID()
  }
  return   `${Date.now()}-${Math.random().toString(16).slice(2)}`
}

function readCartFromStorage() {
  try {
    const raw = localStorage.getItem(CART_STORAGE_KEY)
    if (!raw) return []
    const parsed = JSON.parse(raw)
    if (!Array.isArray(parsed)) return []

    // Migrate old format with quantity to "one row = one position".
    return parsed.flatMap((item) => {
      const quantity = Math.max(1, Number(item.quantity) || 1)
      const base = { ...item }
      delete base.quantity
      return Array.from({ length: quantity }).map(() => ({
        ...base,
        cartItemId: makeCartItemId(),
      }))
    })
  } catch {
    return []
  }
}

function  normalizeCartProduct(product)  {
  const imageUrl = resolveImageUrl(product?.imageUrl ||
product?.image)

  return {
    ...product,
    name: product?.name || product?.title || 'Товар',
    imageUrl,
  }
}

export function CartProvider({ children }) {
  const [items, setItems] = useState(readCartFromStorage)

  useEffect(() => {
    localStorage.setItem(CART_STORAGE_KEY,  JSON.stringify(items))
  }, [items])

  const value = useMemo(() => {
    const addItem = (product) => {
      const normalized = normalizeCartProduct(product)
      setItems((prev) => [...prev, { ...normalized, cartItemId:
makeCartItemId()  }])
    }

    const removeItem = (cartItemId) => {
```

```
      setItems((prev) => prev.filter((item) => item.cartItemId !==
cartItemId))
    }

    const clearCart = () => {
      setItems([])
    }

    const totalItems = items.length
    const totalPrice = items.reduce((sum, item) => sum +
Number(item.price), 0)

    return {
      items,
      totalItems,
      totalPrice,
      addItem,
      removeItem,
      clearCart,
    }
  }, [items])

  return <CartContext.Provider
value={value}>{children}</CartContext.Provider>
}

export function useCart() {
  const context = useContext(CartContext)
  if (!context) {
    throw new Error('useCart must be used within CartProvider')
  }
  return context
}
```

Файл «CartDrawer»

```
import { useEffect, useState } from 'react'
import { createOrder } from '../api/shopApi'
import { useCart } from '../context/CartContext'
import OrderSuccessModal from './OrderSuccessModal'
import ErrorState from './ErrorState'

function CartDrawer({ open, onClose }) {
  const { items, totalPrice, removeItem, clearCart, totalItems } =
useCart()
  const [submitting, setSubmitting] = useState(false)
  const [error, setError] = useState('')
  const [successOpen, setSuccessOpen] = useState(false)
  const [phoneDigits, setPhoneDigits] = useState('')
  const [submittedPhone, setSubmittedPhone] = useState('')
  const [submittedTotalPrice, setSubmittedTotalPrice] = useState(0)
  const [submittedTotalItems, setSubmittedTotalItems] = useState(0)
```

```
  useEffect(() => {
    if (!open) return undefined

    const previousOverflow = document.body.style.overflow
    document.body.style.overflow = 'hidden'

    return () => {
      document.body.style.overflow = previousOverflow
    }
  }, [open])

  const submitOrder = async () => {
    if (!items.length) return
    if (phoneDigits.length !== 10) {
      setError('Введите номер телефона в формате +7XXXXXXXXXX')
      return
    }

    try {
      setSubmitting(true)
      setError('')
      await createOrder({
        items: items.map((item) => ({
          name: item.name,
          price: item.price,
        })),
        totalPrice,
        phone: `+7${phoneDigits}`,
        createdAt: new Date().toISOString(),
      })
      setSubmittedPhone(`+7${phoneDigits}`)
      setSubmittedTotalPrice(totalPrice)
      setSubmittedTotalItems(totalItems)
      clearCart()
      setPhoneDigits('')
      onClose()
      setSuccessOpen(true)
    } catch (requestError) {
      setError(requestError.message || 'Не удалось оформить заказ')
    } finally {
      setSubmitting(false)
    }
  }

  return (
    <>
      <div className={`drawer-overlay ${open ? 'show' : ''}`}
onClick={onClose} />
      <aside className={`cart-drawer ${open ? 'open' : ''}`} aria-
hidden={!open}>
        <div  className="drawer-header">
          <h3>Корзина ({totalItems})</h3>
```

```
          <button  className="drawer-close"  onClick={onClose}>
            ×
          </button>
        </div>

        {error && <ErrorState text={error} />}

        {!items.length ? (
          <p  className="muted">Корзина  пустая.</p>
        ) : (
          <>
            <div  className="cart-list">
              {items.map((item) => (
                <article className="cart-item"
key={item.cartItemId}>
                  <div  className="cart-item-main">
                    <img
                      className="cart-item-image"
                      src={item.imageUrl ||
'https://placehold.co/96x96?text=Item'}
                      alt={item.name || 'Товар'}
                      onError={(event) => {
                        event.currentTarget.src =
'https://placehold.co/96x96?text=Item'
                      }}
                    />
                    <div>
                    <h3>{item.name}</h3>
                    <p
className="muted">{Number(item.price).toLocaleString('ru-RU')}
руб.</p>
                    </div>
                  </div>
                  <div  className="cart-actions">
                    <button className="remove-btn" onClick={() =>
removeItem(item.cartItemId)}>
                      Удалить
                    </button>
                  </div>
                </article>
              ))}
            </div>

            <div  className="cart-footer">
              <strong>Итого:   {totalPrice.toLocaleString('ru-RU')}
руб.</strong>
            </div>
            <div  className="cart-phone">
              <label  className="cart-phone-label"  htmlFor="cart-
phone-input">
                Телефон для заказа
              </label>
```

```
              <div  className="phone-input-row">
                <span  className="phone-prefix">+7</span>
                <input
                  id="cart-phone-input"
                  type="tel"
                  inputMode="numeric"
                  autoComplete="tel"
                  value={phoneDigits}
                  onChange={(event) => {
                    const digitsOnly =
event.target.value.replace(/\D/g,  '').slice(0,  10)
                    setPhoneDigits(digitsOnly)
                  }}
                  placeholder="XXXXXXXXXX"
                />
              </div>
            </div>
            <div  className="cart-footer">
              <button className="add-btn" onClick={submitOrder}
disabled={submitting}>
                {submitting ? 'Отправка...' : 'Оформить заказ'}
              </button>
            </div>
          </>
        )}
      </aside>

      <OrderSuccessModal
        open={successOpen}
        onClose={() => setSuccessOpen(false)}
        phone={submittedPhone}
        totalPrice={submittedTotalPrice}
        totalItems={submittedTotalItems}
      />
    </>
  )
}

export default CartDrawer

import { useEffect } from 'react'

function OrderSuccessModal({ open, onClose, phone, totalPrice,
totalItems }) {
  useEffect(() => {
    if (!open) return undefined
    const handleKeyDown = (event) => {
      if (event.key === 'Escape') onClose()
    }
    window.addEventListener('keydown', handleKeyDown)
    return () => window.removeEventListener('keydown',
handleKeyDown)
```

```
  }, [open, onClose])

  if (!open) return null

  const formattedDate = new Date().toLocaleString('ru-RU')

  return (
    <div className="modal-overlay" role="presentation"
onClick={onClose}>
      <div
        className="modal-card success-modal"
        role="dialog"
        aria-modal="true"
        aria-label="Заказ  оформлен"
        onClick={(event) => event.stopPropagation()}
      >
        <div  className="success-modal-head">
          <div  className="success-modal-icon"  aria-hidden="true">
            ✓
          </div>
          <div>
            <h3>Спасибо за заказ!</h3>
            <p className="success-modal-subtitle">
              Заказ принят в обработку.
            </p>
          </div>
        </div>

        <p  className="success-modal-text">
          По заказу с вами свяжутся по номеру телефона
<strong>{phone || '+7XXXXXXXXXX'}</strong>.
        </p>

        <div  className="success-modal-info">
          <div  className="success-modal-row">
            <span>Количество  позиций</span>
            <strong>{totalItems || 0}</strong>
          </div>
          <div  className="success-modal-row">
            <span>Сумма заказа</span>
            <strong>{Number(totalPrice  ||  0).toLocaleString('ru-
RU')} руб.</strong>
          </div>
          <div  className="success-modal-row">
            <span>Дата оформления</span>
            <strong>{formattedDate}</strong>
          </div>
        </div>

        <div  className="modal-actions">
          <button className="add-btn modal-ok-btn"
onClick={onClose}>
```

```
            Хорошо
          </button>
        </div>
      </div>
    </div>
  )
}

export default OrderSuccessModal
```

```
package sportswear.sportswear.core.accessories.service;

import lombok.AllArgsConstructor;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
import sportswear.sportswear.core.accessories.domain.Accessories;
import sportswear.sportswear.core.accessories.rep.AccessoriesRep;
import  sportswear.sportswear.core.accessories.view.AccessoriesView;
import sportswear.sportswear.core.shoes.domain.Shoes;

import java.util.List;

@Service
@AllArgsConstructor
public class AccessoriesService {

    private final AccessoriesRep accessoriesRep;

    public AccessoriesView getAccessories(int page, int size,
Integer sizeProduct, String type, String colour, String sortBy,
String direction) {
        Sort sort = Sort.unsorted();
        if (sortBy != null) {
            sort = direction != null &&
direction.equalsIgnoreCase("desc")
                    ? Sort.by(sortBy).descending()
                    : Sort.by(sortBy).ascending();
        }

        Pageable pageable = PageRequest.of(page, size, sort);

        Specification<Accessories> spec = Specification.where(null);

        if (type != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("type"), type));
        }
        if (colour != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("colour"), colour));
```

```
        }
        if (sizeProduct != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("sizeBall"), sizeProduct));
        }

        AccessoriesView accessoriesView = new AccessoriesView();
        accessoriesView.setItems(accessoriesRep.findAll(spec,
pageable).getContent());
        accessoriesView.setCount(accessoriesRep.findAll(spec,
pageable).getTotalElements());

        return accessoriesView;
    }

    public Accessories getAccessoriesById(Long id) {

        return  accessoriesRep.findById(id).orElse(null);

    }

}
package sportswear.sportswear.core.accessories.web;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import sportswear.sportswear.core.accessories.domain.Accessories;
import
sportswear.sportswear.core.accessories.service.AccessoriesService;
import  sportswear.sportswear.core.accessories.view.AccessoriesView;
import sportswear.sportswear.core.shoes.domain.Shoes;
import  sportswear.sportswear.core.shoes.service.ShoesService;

import java.util.List;

@RestController
@RequestMapping("/api")
@Tag(name="accessories")
@Validated
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:3333")
public class AccessoriesWeb {

    private final AccessoriesService accessoriesService;

    @GetMapping("/getAccessories")
    @ResponseBody
    @Operation(
            summary = "Получение списка аксессуаров",
```

```
            description = "Позволяет получить список аксессуаров"
    )
    public AccessoriesView getAccessories(@RequestParam(required =
false, defaultValue = "10") int size,
                                          @RequestParam(required =
false, defaultValue = "0") int page, @RequestParam(required = false)
Integer sizeShoes,
                                          @RequestParam(required =
false) String type, @RequestParam(required = false) String colour,
                                          @RequestParam(required =
false) String material, @RequestParam(required = false) String
sortBy,
                                          @RequestParam(required  =
false) String direction){
        return accessoriesService.getAccessories(page, size,
sizeShoes, type, colour, sortBy, direction);
    }

    @GetMapping("/getAccessories/{id}")
    @ResponseBody
    @Operation(
            summary = "Получение аксессуаров по id",
            description = "Позволяет получить информацию об
аксессуаре по id"
    )
    public Accessories getAccessoriesById(@PathVariable Long id) {
        return accessoriesService.getAccessoriesById(id);
    }

}
package sportswear.sportswear.core.accessories.domain;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import jakarta.validation.constraints.NotNull;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Table(name= "accessories")
public class Accessories {

        @Id
        private Long id;

        @NotNull
```

```
        private String name;

        @NotNull
        private String type;

        @NotNull
        private Double price;

        @NotNull
        private String imageUrl;

        @NotNull
        private String sizeProduct;

        @NotNull
        private String colour;

        @NotNull
        private String description;

}
package sportswear.sportswear.core.menu.service;

import    org.springframework.transaction.annotation.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import   sportswear.sportswear.core.menu.domain.MenuItem;
import   sportswear.sportswear.core.menu.rep.MenuItemRep;
import sportswear.sportswear.core.menu.web.MenuView;

import java.util.List;

@Service
@RequiredArgsConstructor
public class MenuService {

    private final MenuItemRep menuItemRep;

    @Transactional(readOnly = true)
    public List<MenuView> getMenu() {
        return menuItemRep.findByParentIsNull()
                .stream()
                .map(this::toView)
                .toList();
    }

    private MenuView toView(MenuItem entity) {

        MenuView view = new MenuView();
        view.setId(entity.getId());
        view.setTitle(entity.getTitle());
        view.setPath(entity.getPath());
        view.setPageId(entity.getPageId());
```

```
        if (entity.getChildren() != null &&
!entity.getChildren().isEmpty())  {
            view.setChildren(
                    entity.getChildren()
                            .stream()
                            .map(this::toView)
                            .toList()
            );
        }

        return view;
    }

}
```

```
package sportswear.sportswear.core.menu.web;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import sportswear.sportswear.core.menu.service.MenuService;

import java.util.List;

@RestController
@RequestMapping("/api")
@Tag(name="menu")
@Validated
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:3333")
public class MenuController {

    private static final Logger logger =
LoggerFactory.getLogger(MenuController.class);
    private final MenuService menuService;

    @GetMapping("/menu")
    @ResponseBody
    @Operation(
            summary = "Получение бокового меню",
            description = "Позволяет получить боковое меню"
    )
    public List<MenuView> getPageById() {
        logger.info("Получен запрос на получение меню");
        return menuService.getMenu();
    }

}
```

```
package sportswear.sportswear.core.menu.web;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import sportswear.sportswear.core.menu.domain.MenuItem;

import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class MenuView {

    private Long id;

    private String title;

    private String path;

    private Long pageId;

    private List<MenuView> children = new ArrayList<>();

}
package sportswear.sportswear.core.menu.domain;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.util.ArrayList;
import java.util.List;

@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Table(name= "menu")
public class MenuItem {

    @Id
    private Long id;

    private String title;
```

```
    private String path;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "parent_id")
    private MenuItem parent;

    private Long pageId;

    @OneToMany(mappedBy = "parent", cascade = CascadeType.ALL)
    private List<MenuItem> children = new ArrayList<>();

}



package sportswear.sportswear.core.page.service;

import jakarta.persistence.EntityNotFoundException;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import sportswear.sportswear.core.page.domain.Page;
import sportswear.sportswear.core.page.rep.PageRep;
import  sportswear.sportswear.core.page.web.PageView;
import sportswear.sportswear.core.product.domain.Product;
import  sportswear.sportswear.core.product.view.ProductView;

import java.util.List;

@Service
@RequiredArgsConstructor
public class PageService {

    private final PageRep pageRep;

    public Page getBookById(Long id) {
        return pageRep.findById(id)
                .orElseThrow(() -> new EntityNotFoundException("Page
with id " + id + " not found"));
    }

    public PageView getBook(Long id) {

       return toView (getBookById(id));
    }

    public static PageView toView(Page page) {
        if (page == null) return null;

        PageView view = new PageView();
        view.setId(page.getId());
        view.setPageName(page.getPageName());
        view.setContactInfo(page.getContactInfo());
        view.setFilterParam(page.getFilterParam());
```

```
        if (page.getProducts() != null) {
            List<ProductView> products = page.getProducts().stream()
                    .map(PageService::toView)
                    .toList();
            view.setProducts(products);
        }

        return view;
    }

    private static ProductView toView(Product product) {
        if (product == null) return null;

        ProductView view = new ProductView();
        view.setId(product.getId());
        view.setName(product.getName());
        view.setPrice(product.getPrice());
        view.setPath(product.getPath());
        view.setImageUrl(product.getImageUrl());
        return view;
    }

}
```

```
package sportswear.sportswear.core.page.web;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import sportswear.sportswear.core.page.service.PageService;

@RestController
@RequestMapping("/api")
@Tag(name="page")
@Validated
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:3333")
public class PageController {

    private static final Logger logger =
LoggerFactory.getLogger(PageController.class);
    private final PageService pageService;

    @GetMapping("/page/{id}")
    @ResponseBody
    @Operation(
            summary = "Получение страницы по id",
            description = "Позволяет найти страницу по id"
```

```
    )
    public PageView getPageById(@PathVariable Long id) {
        logger.info("Получен запрос на получение страницы с id: {}",
id);
        return pageService.getBook(id);
    }

}
package sportswear.sportswear.core.page.web;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import  sportswear.sportswear.core.contactinfo.domain.ContactInfo;
import sportswear.sportswear.core.filter.domain.FilterParam;
import sportswear.sportswear.core.product.domain.Product;
import  sportswear.sportswear.core.product.view.ProductView;

import java.util.List;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class PageView {

    private Long id;
    private String pageName;
    private List<ProductView> products;
    private ContactInfo contactInfo;
    private FilterParam filterParam;

}
package sportswear.sportswear.core.ball.service;

import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
import  sportswear.sportswear.core.accessories.view.AccessoriesView;
import sportswear.sportswear.core.ball.domain.Ball;
import sportswear.sportswear.core.ball.rep.BallRep;
import org.springframework.data.domain.Sort;
import sportswear.sportswear.core.ball.view.BallView;

import java.util.List;

@Service
@RequiredArgsConstructor
public class BallService {
```

```
    private final BallRep ballRep;

    public BallView getBall(int page, int size, Integer sizeBall,
String type, String colour, String material, String sortBy, String
direction) {

        Sort sort = Sort.unsorted();

        if (sortBy != null) {
            sort = direction != null &&
direction.equalsIgnoreCase("desc")
                    ? Sort.by(sortBy).descending()
                    : Sort.by(sortBy).ascending();
        }

        Pageable pageable = PageRequest.of(page, size, sort);

        Specification<Ball> spec = Specification.where(null);

        if (type != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("type"), type));
        }
        if (colour != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("colour"), colour));
        }
        if (material != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("material"), material));
        }
        if (sizeBall != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("sizeBall"), sizeBall));
        }
        BallView ballView = new BallView();
        ballView.setItems(ballRep.findAll(spec,
pageable).getContent());
        ballView.setCount(ballRep.findAll(spec,
pageable).getTotalElements());

        return ballView;
    }

    public Ball getBallById(Long id) {

        return ballRep.findById(id).orElse(null);

    }

}
```

```
package sportswear.sportswear.core.ball.view;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import sportswear.sportswear.core.accessories.baseview.BaseView;
import  sportswear.sportswear.core.accessories.domain.Accessories;
import sportswear.sportswear.core.ball.domain.Ball;

import java.util.List;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class BallView extends BaseView {

    private List<Ball> items;

}
package sportswear.sportswear.core.ball.web;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import sportswear.sportswear.core.ball.domain.Ball;
import sportswear.sportswear.core.ball.service.BallService;
import sportswear.sportswear.core.ball.view.BallView;

import java.util.List;

@RestController
@RequestMapping("/api")
@Tag(name = "ball")
@Validated
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:3333")
public class BallController {

    private final BallService ballService;

    @GetMapping("/ball")
    @ResponseBody
    @Operation(
            summary = "Получение списка мячей",
            description = "Позволяет получить список мячей"
    )
    public BallView getBall(@RequestParam(required = false,
defaultValue = "10") int size, @RequestParam(required = false,
defaultValue = "0") int page, @RequestParam(required = false)
```

```
Integer sizeBall, @RequestParam(required = false) String type,
@RequestParam(required = false) String colour,
@RequestParam(required = false) String material,
@RequestParam(required = false) String sortBy,
@RequestParam(required = false) String direction) {
        return ballService.getBall(page, size, sizeBall, type,
colour, material, sortBy, direction);
    }

    @GetMapping("/ball/{id}")
    @ResponseBody
    @Operation(
            summary = "Получение объекта мяч по id",
            description = "Позволяет получить информацию о мяче по
id"
    )
    public Ball getBallById(@PathVariable Long id) {
        return ballService.getBallById(id);
    }

}
package sportswear.sportswear.core.gloves.service;

import lombok.AllArgsConstructor;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
import  sportswear.sportswear.core.accessories.view.AccessoriesView;
import sportswear.sportswear.core.gloves.rep.GlovesRep;
import sportswear.sportswear.core.gloves.domain.Gloves;
import  sportswear.sportswear.core.gloves.view.GlovesView;

import java.util.List;

@Service
@AllArgsConstructor
public class GlovesService {

    private final GlovesRep glovesRep;

    public GlovesView getGloves(int page, int size,
                                String type, String colour,
                                String sizeGloves,
                                String sortBy, String direction) {

        Sort sort = Sort.unsorted();
        if (sortBy != null) {
            sort = direction != null &&
direction.equalsIgnoreCase("desc")
                    ? Sort.by(sortBy).descending()
                    : Sort.by(sortBy).ascending();
```

```
        }

        Pageable pageable = PageRequest.of(page, size, sort);
        Specification<Gloves> spec = Specification.where(null);

        if (type != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("type"), type));
        }
        if (colour != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("colour"), colour));
        }
        if (sizeGloves != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("sizeGloves"), sizeGloves));
        }

        GlovesView glovesView = new GlovesView();
        glovesView.setItems(glovesRep.findAll(spec,
pageable).getContent());
        glovesView.setCount(glovesRep.findAll(spec,
pageable).getTotalElements());

        return glovesView;
    }

    public Gloves getById(Long id) {
        return glovesRep.findById(id).orElse(null);
    }
}
package sportswear.sportswear.core.gloves.web;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import sportswear.sportswear.core.gloves.domain.Gloves;
import sportswear.sportswear.core.gloves.service.GlovesService;
import sportswear.sportswear.core.gloves.view.GlovesView;

import java.util.List;

@RestController
@RequestMapping("/api")
@Tag(name = "gloves")
@Validated
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:3333")
public class GlovesController {

    private final GlovesService glovesService;
```

```
    @GetMapping("/getGloves")
    @ResponseBody
    @Operation(
            summary = "Получение списка перчаток",
            description = "Позволяет получить список перчаток"
    )
    public GlovesView getGloves(
            @RequestParam(required = false, defaultValue = "10") int
size,
            @RequestParam(required = false, defaultValue = "0") int
page,
            @RequestParam(required = false) String type,
            @RequestParam(required = false) String colour,
            @RequestParam(required = false) String sizeGloves,
            @RequestParam(required = false) String sortBy,
            @RequestParam(required = false) String direction) {

        return glovesService.getGloves(page, size, type, colour,
sizeGloves, sortBy, direction);
    }

    @GetMapping("/getGloves/{id}")
    @ResponseBody
    @Operation(
            summary = "Получение перчаток по id",
            description = "Позволяет получить информацию о перчатках
по id"
    )
    public Gloves getById(@PathVariable Long id) {
        return glovesService.getById(id);
    }
}
package sportswear.sportswear.core.jacket.service;

import lombok.AllArgsConstructor;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
import  sportswear.sportswear.core.accessories.view.AccessoriesView;
import sportswear.sportswear.core.jacket.domain.Jacket;
import sportswear.sportswear.core.jacket.rep.JacketRep;
import  sportswear.sportswear.core.jacket.view.JacketView;

import java.util.List;

@Service
@AllArgsConstructor
public class JacketService {

    private final JacketRep jacketRep;
```

```
    public JacketView getJackets(int page, int size,
                                 String sizeJacket,
                                 String colour,
                                 String material,
                                 String sortBy,
                                 String direction) {

        Sort sort = Sort.unsorted();
        if (sortBy != null) {
            sort = direction != null &&
direction.equalsIgnoreCase("desc")
                    ? Sort.by(sortBy).descending()
                    : Sort.by(sortBy).ascending();
        }

        Pageable pageable = PageRequest.of(page, size, sort);

        Specification<Jacket> spec = Specification.where(null);

        if (colour != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("colour"), colour));
        }

        if (material != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("material"), material));
        }

        if (sizeJacket != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("sizeJacket"), sizeJacket));
        }

        JacketView jacketView = new JacketView();
        jacketView.setItems(jacketRep.findAll(spec,
pageable).getContent());
        jacketView.setCount(jacketRep.findAll(spec,
pageable).getTotalElements());

        return jacketView;
    }

    public Jacket getJacketById(Long id) {
        return jacketRep.findById(id).orElse(null);
    }
}
package sportswear.sportswear.core.jacket.web;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;
```

```
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import sportswear.sportswear.core.jacket.domain.Jacket;
import sportswear.sportswear.core.jacket.service.JacketService;
import sportswear.sportswear.core.jacket.view.JacketView;

import java.util.List;

@RestController
@RequestMapping("/api")
@Tag(name = "jacket")
@Validated
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:3333")
public class JacketController {

    private final JacketService jacketService;

    @GetMapping("/jackets")
    @ResponseBody
    @Operation(
            summary = "Получение списка курток",
            description = "Возвращает список курток с возможностью
фильтрации по размеру, цвету, материалу и сортировки"
    )
    public JacketView getJackets(
            @RequestParam(required = false, defaultValue = "10") int
size,
            @RequestParam(required = false, defaultValue = "0") int
page,
            @RequestParam(required = false) String sizeJacket,
            @RequestParam(required = false) String colour,
            @RequestParam(required = false) String material,
            @RequestParam(required = false) String sortBy,
            @RequestParam(required = false) String direction) {

        return jacketService.getJackets(page, size, sizeJacket,
colour, material, sortBy, direction);
    }

    @GetMapping("/jackets/{id}")
    @ResponseBody
    @Operation(
            summary = "Получение куртки по ID",
            description = "Возвращает информацию о конкретной куртке
по её идентификатору"
    )
    public Jacket getJacketById(@PathVariable Long id) {
        return jacketService.getJacketById(id);
    }
}
package sportswear.sportswear.core.order.service;
```

```
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import sportswear.sportswear.core.order.domain.Order;
import   sportswear.sportswear.core.order.rep.OrderRep;

@Service
@RequiredArgsConstructor
public class OrderService {

    private final OrderRep orderRep;

    @Transactional
    public Order saveOrder(Order order) {

        return orderRep.save(order);
    }

}

package sportswear.sportswear.core.order.web;

import lombok.AllArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import  org.springframework.web.bind.annotation.*;
import sportswear.sportswear.core.order.domain.Order;
import  sportswear.sportswear.core.order.service.OrderService;

import java.util.Map;

@RestController
@RequestMapping("/api")
@AllArgsConstructor
@CrossOrigin(origins = "http://localhost:3333")
public class OrderController {

    private final OrderService orderService;

    @PostMapping("/order")
    public  ResponseEntity<Void>  saveOrders(@RequestBody  Order  order)
{
        orderService.saveOrder(order);
        return ResponseEntity.status(HttpStatus.CREATED).build();
    }
}
package sportswear.sportswear.core.order.web;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
```

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class OrderView {

    private Long price;

    private String name;

}
```

```
package sportswear.sportswear.core.shoes.service;

import lombok.AllArgsConstructor;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
import  sportswear.sportswear.core.accessories.view.AccessoriesView;
import sportswear.sportswear.core.ball.domain.Ball;
import sportswear.sportswear.core.inventory.domain.Inventory;
import sportswear.sportswear.core.shoes.domain.Shoes;
import sportswear.sportswear.core.shoes.rep.ShoesRep;
import  sportswear.sportswear.core.shoes.view.ShoesView;

import java.util.List;

@Service
@AllArgsConstructor
public class ShoesService {

    private final ShoesRep shoesRep;

    public ShoesView getShoes(int page, int size, String sizeShoes,
String type, String colour, String material, String sortBy, String
direction) {
        Sort sort = Sort.unsorted();
        if (sortBy != null) {
            sort = direction != null &&
direction.equalsIgnoreCase("desc")
                    ? Sort.by(sortBy).descending()
                    : Sort.by(sortBy).ascending();
        }

        Pageable pageable = PageRequest.of(page, size, sort);

        Specification<Shoes> spec = Specification.where(null);

        if (type != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("type"), type));
```

```
        }
        if (colour != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("colour"), colour));
        }
        if (material != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("material"), material));
        }
        if (sizeShoes != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("sizeShoes"), sizeShoes));
        }

        ShoesView shoesView = new ShoesView();
        shoesView.setItems(shoesRep.findAll(spec,
pageable).getContent());
        shoesView.setCount(shoesRep.findAll(spec,
pageable).getTotalElements());

        return shoesView;
    }

    public Shoes getShoesById(Long id) {

        return shoesRep.findById(id).orElse(null);

    }

}
```

```
package sportswear.sportswear.core.shoes.web;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import sportswear.sportswear.core.shoes.domain.Shoes;
import sportswear.sportswear.core.shoes.service.ShoesService;
import sportswear.sportswear.core.shoes.view.ShoesView;

import java.util.List;

@RestController
@RequestMapping("/api")
@Tag(name="shoes")
@Validated
@RequiredArgsConstructor
@CrossOrigin(origins = "http://localhost:3333")
public class ShoesController {
```

```
    private final ShoesService shoesService;

    @GetMapping("/getShoes")
    @ResponseBody
    @Operation(
            summary = "Получение списка обуви",
            description = "Позволяет получить список обуви"
    )
    public ShoesView getShoes(@RequestParam(required = false,
defaultValue = "10") int size, @RequestParam(required = false,
defaultValue = "0") int page, @RequestParam(required = false) String
sizeShoes, @RequestParam(required = false) String type,
@RequestParam(required = false) String colour,
@RequestParam(required = false) String material,
@RequestParam(required = false) String sortBy,
@RequestParam(required = false) String direction){
        return shoesService.getShoes(page, size, sizeShoes, type,
colour, material, sortBy, direction);
    }

    @GetMapping("/getShoes/{id}")
    @ResponseBody
    @Operation(
            summary = "Получение обуви по id",
            description = "Позволяет получить информацию об обуви по
id"
    )
    public Shoes getShoesById(@PathVariable Long id) {
        return shoesService.getShoesById(id);
    }

}

package sportswear.sportswear.core.sweater.service;

import lombok.AllArgsConstructor;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
import  sportswear.sportswear.core.accessories.view.AccessoriesView;
import sportswear.sportswear.core.sweater.domain.Sweater;
import sportswear.sportswear.core.sweater.rep.SweaterRep;
import  sportswear.sportswear.core.sweater.view.SweaterView;

import java.util.List;

@Service
@AllArgsConstructor
public class SweaterService {
```

```
    private final SweaterRep sweaterRep;

    public SweaterView getSweaters(int page, int size,
                                   String sizeSweater,
                                   String colour,
                                   String material,
                                   String sortBy,
                                   String direction) {

        Sort sort = Sort.unsorted();
        if (sortBy != null) {
            sort = direction != null &&
direction.equalsIgnoreCase("desc")
                    ? Sort.by(sortBy).descending()
                    : Sort.by(sortBy).ascending();
        }

        Pageable pageable = PageRequest.of(page, size, sort);

        Specification<Sweater> spec = Specification.where(null);

        if (colour != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("colour"), colour));
        }

        if (material != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("material"), material));
        }

        if (sizeSweater != null) {
            spec = spec.and((root, query, cb) ->
cb.equal(root.get("sizeSweater"),  sizeSweater));
        }

        SweaterView sweaterView = new SweaterView();
        sweaterView.setItems(sweaterRep.findAll(spec,
pageable).getContent());
        sweaterView.setCount(sweaterRep.findAll(spec,
pageable).getTotalElements());

        return sweaterView;
    }

    public Sweater getSweaterById(Long id) {
        return sweaterRep.findById(id).orElse(null);
    }
}
```

Приложение В

(обязательное)

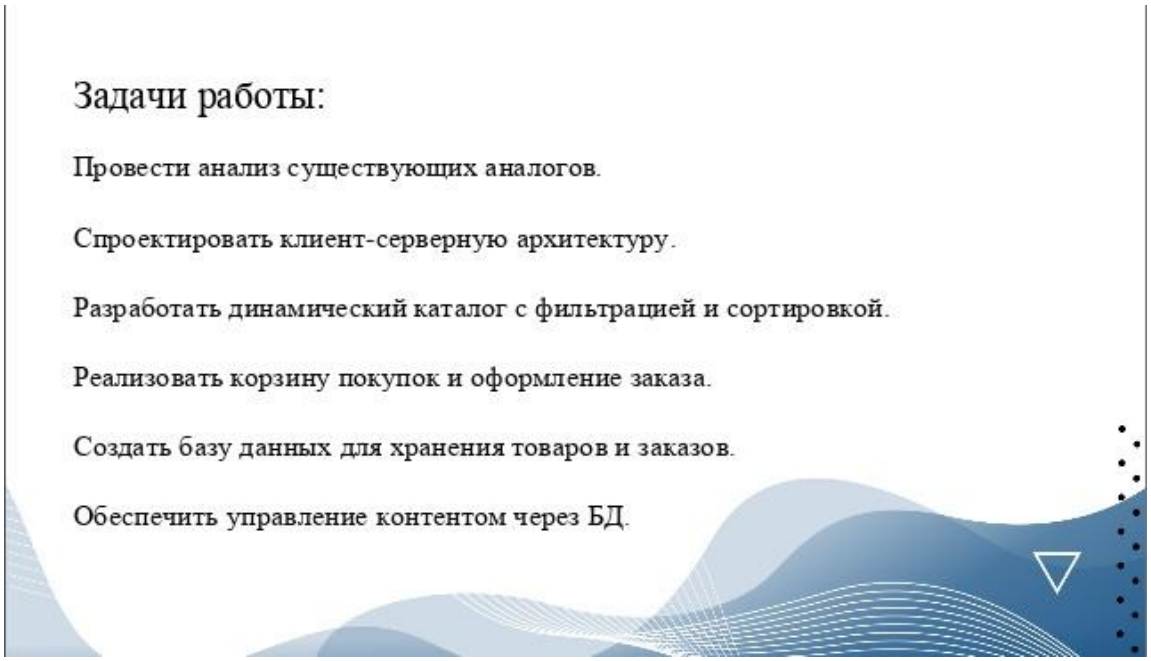
WEB приложение – магазин футбольной экипировки «Sportwear Store»

Презентация

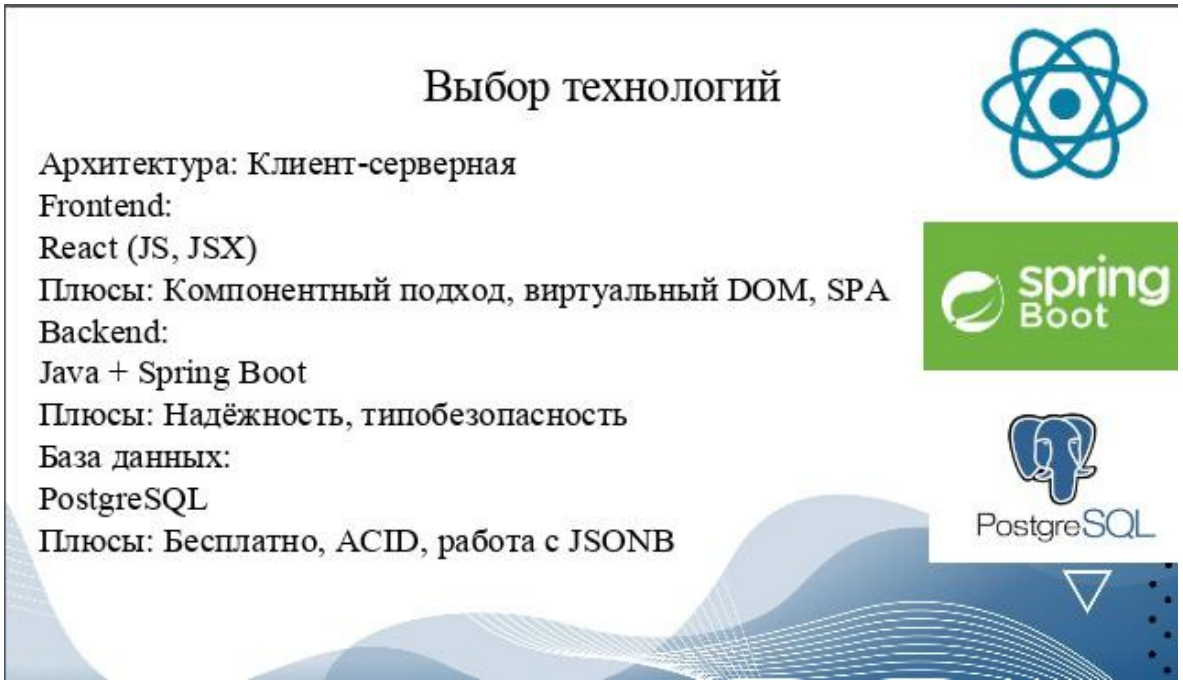
Слайд 1:

Слайд 2:

Слайд 3:



Слайд 4:



Слайд 5:

Архитектура приложения (MVC)

Паттерн MVC:

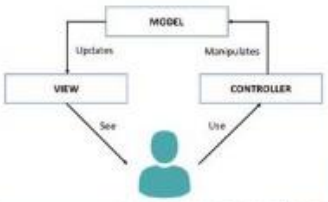
Model (Модель): Классы сущностей (Ball, Shoes, Order), репозитории Spring Data JPA.

View (Представление): React-компоненты + CSS.

Controller (Контроллер): @RestController (обработка HTTP-запросов).

Схема взаимодействия:

Браузер (React) → HTTP/JSON → Контроллер (Spring) → Сервис → Репозиторий → PostgreSQL



Слайд 6:

Проектирование базы данных

ER-диаграмма:

page (страницы каталога)

product (общие товары)

ball, shoes, jacket, accessories (детальные таблицы категорий)

filter_param (хранение фильтров в JSONB)

orders (заказы)

Слайд 7:

Реализация клиентской части (React)

Функциональность:

Маршрутизация: React Router (страницы каталога /page/:id, карточка товара /product/:id).

Управление состоянием: Context API (корзина).

HTTP-клиент: Axios (запросы к серверу).

Компоненты: PageItem.jsx, ProductPage.jsx, CartDrawer.jsx.

Особенность: Адаптивная вёрстка и сохранение корзины в localStorage.

Слайд 8:

Реализация серверной части (Spring Boot)

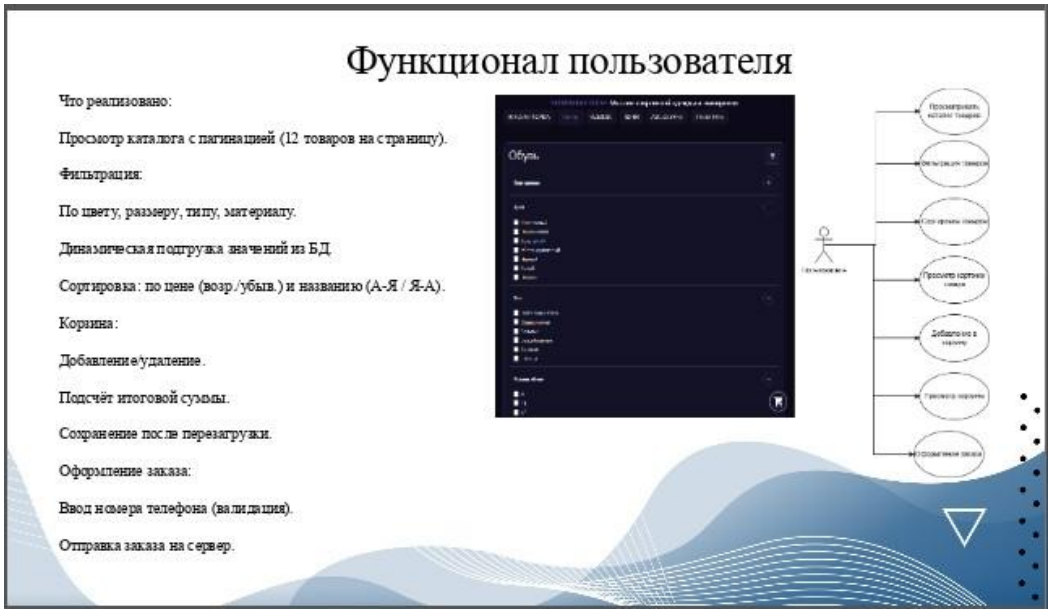
Слои приложения:

Контроллер (BallController.java):
@GetMapping("/ball") → принимает параметры фильтрации.

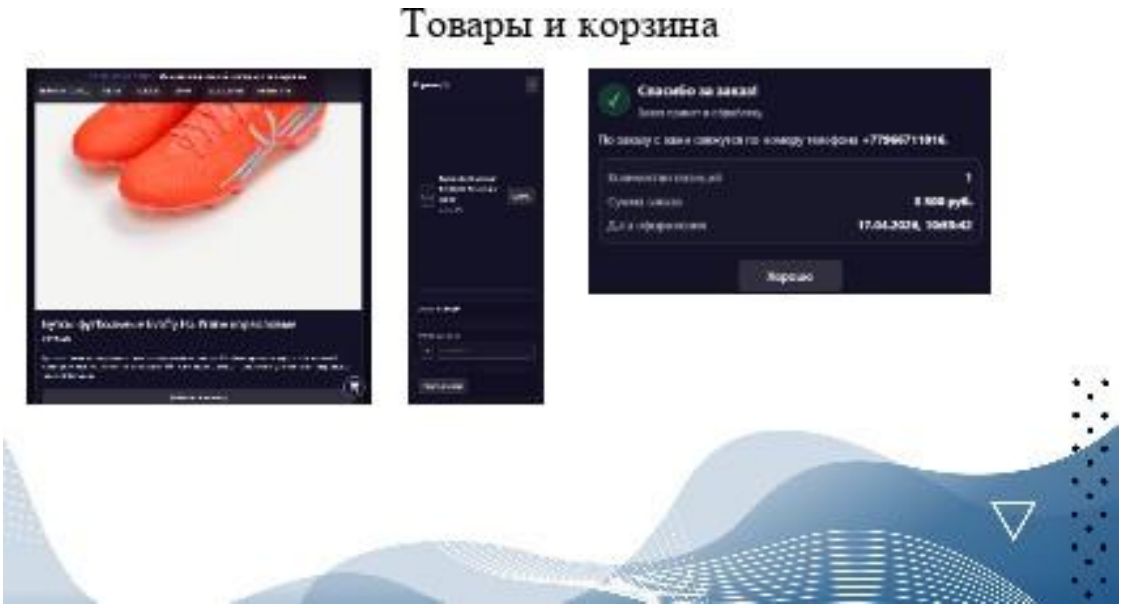
Сервис (BallService.java):
Строит Specification и Pageable, вызывает репозиторий.

Репозиторий (BallRepository.java):
Наследуется от JpaSpecificationExecutor.

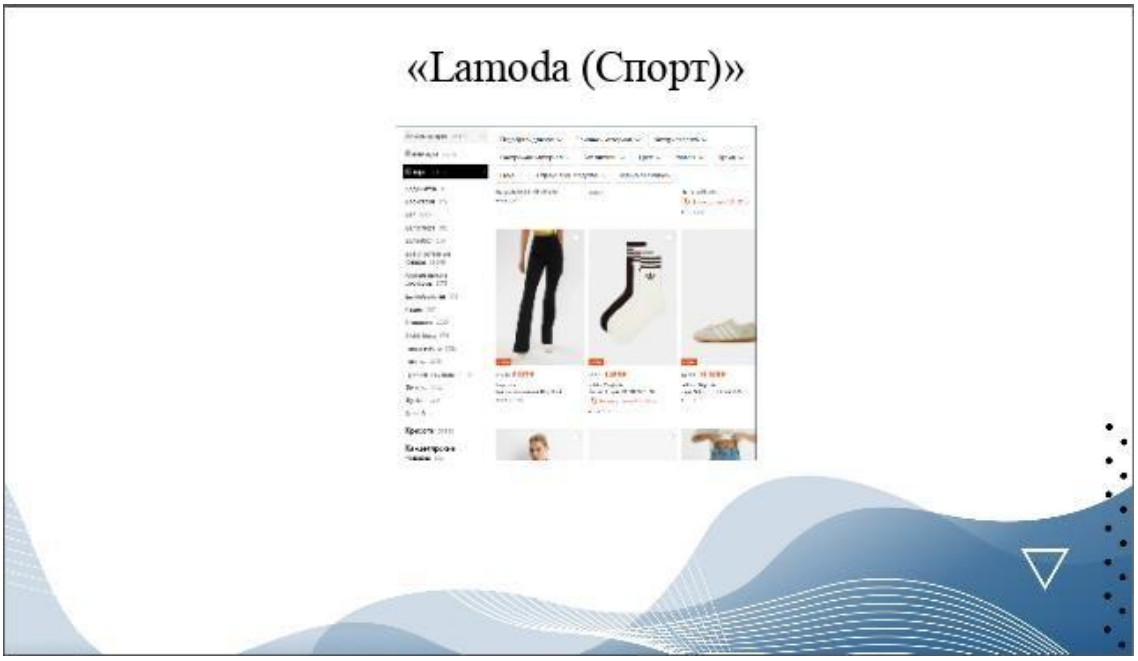
Слайд 9:



Слайд 10:

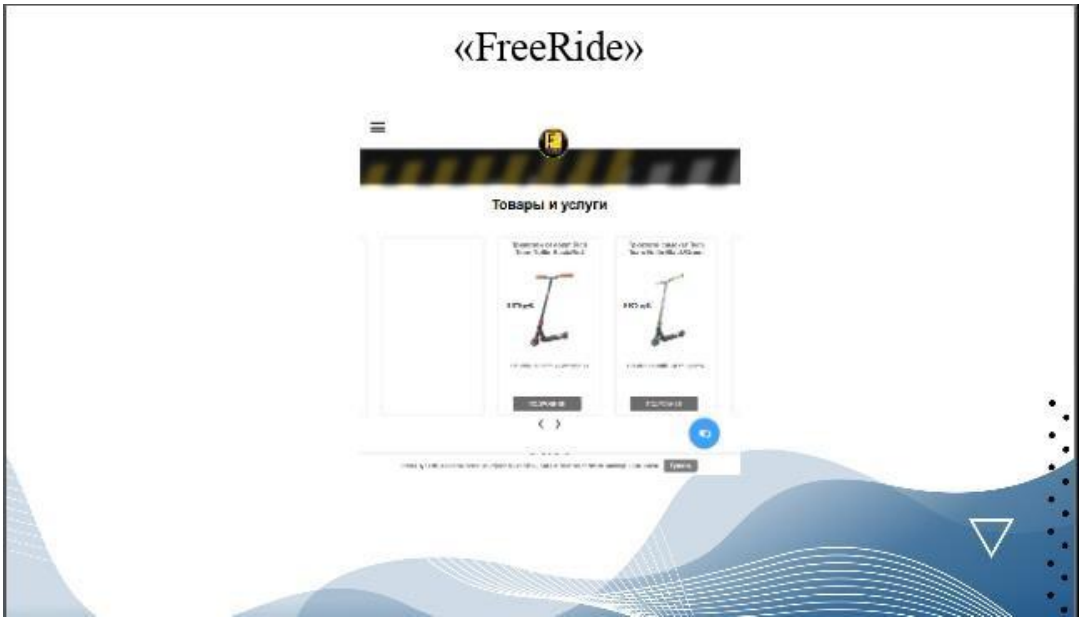


Слайд 11:

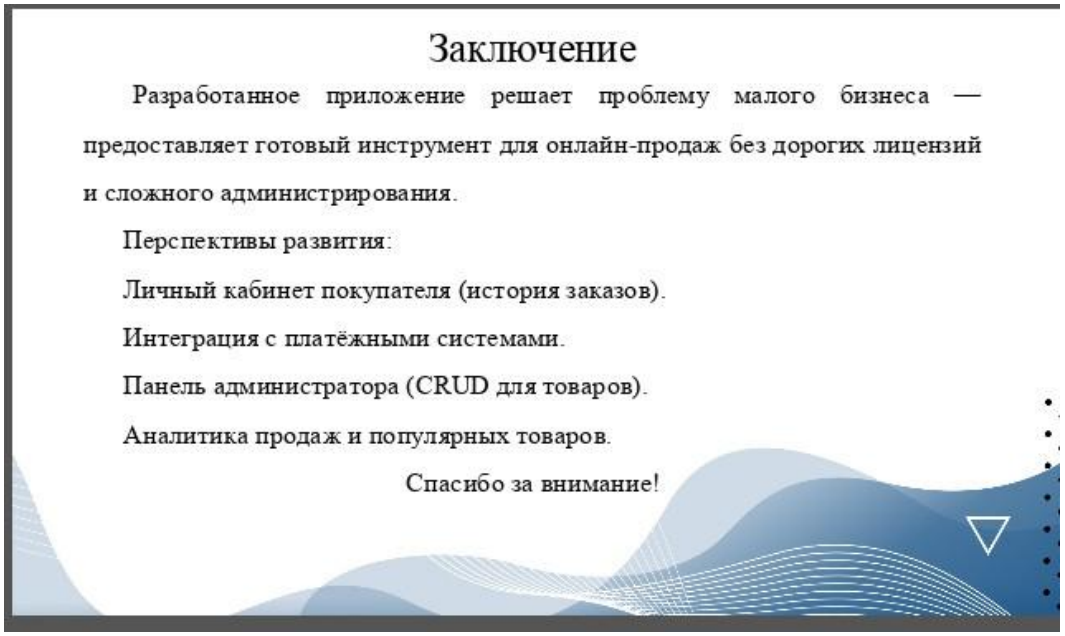


Слайд 12:

Слайд 13:


«FreeRide»
Товары и услуги

Слайд 14:


Заключение
Разработанное приложение решает проблему малого бизнеса — предоставляет готовый инструмент для онлайн-продаж без дорогих лицензий и сложного администрирования.
Перспективы развития:
Личный кабинет покупателя (история заказов).
Интеграция с платёжными системами.
Панель администратора (CRUD для товаров).
Аналитика продаж и популярных товаров.
Спасибо за внимание!