**Содержание**

[1. Постановка задачи 4](#_Toc219425475)

[1.1 Обзор аналогичных решений 4](#_Toc219425476)

[1.2 Техническое задание 7](#_Toc219425477)

[1.3 Выбор средств реализации программного продукта 9](#_Toc219425478)

[1.4 Выводы по разделу 10](#_Toc219425479)

[2 Проектирование страниц веб-сайта 11](#_Toc219425480)

[2.1 Общая структура 11](#_Toc219425481)

[2.2 Выбор способа верстки 15](#_Toc219425482)

[2.3 Выбор стилевого оформления 16](#_Toc219425483)

[2.4 Выбор шрифтового оформления 16](#_Toc219425484)

[2.5 Разработка логотипа 16](#_Toc219425485)

[2.6 Разработка пользовательских элементов 16](#_Toc219425486)

[2.7 Разработка спецэффектов 17](#_Toc219425487)

[2.8 Выводы по разделу 17](#_Toc219425488)

[3 Реализация структуры веб-сайта 18](#_Toc219425489)

[3.1 Структура HTML-документа 18](#_Toc219425490)

[3.2 Структура JSX-файлов 18](#_Toc219425491)

[3.3 Добавление таблицы стилей CSS 18](#_Toc219425492)

[3.4 Выводы по разделу 19](#_Toc219425493)

[4 Тестирование веб-сайта 20](#_Toc219425494)

[4.1 Адаптивный дизайн веб-сайта 20](#_Toc219425495)

[4.2 Кроссбраузерность веб-сайта 20](#_Toc219425496)

[4.3 Выводы по разделу 20](#_Toc219425497)

[5.1 Руководство для пользователя 21](#_Toc219425498)

[5.2 Выводы по разделу 21](#_Toc219425499)

[Заключение 22](#_Toc219425500)

[Список использованных литературных источников 23](#_Toc219425501)

[ПРИЛОЖЕНИЕ А. Макеты веб-страниц 24](#_Toc219425502)

[ПРИЛОЖЕНИЕ Б. Листинг Html документов 30](#_Toc219425503)

[ПРИЛОЖЕНИЕ В. Листинг js/jsx документов 31](#_Toc219425504)

[ПРИЛОЖЕНИЕ Г. Листинг css документов 109](#_Toc219425505)

**Введение**

Темой проекта является интернет-магазин продажи аниме атрибутики. Аниме атрибутика представляет собой книги, статуэтки, брелки, значки и др.

Тема является актуальной на данный момент, также останется такой на многие поколения вперёд. Рынок аниме стремительно развивается, каждый раз преподнося новые и порой интересные тайтлы. В данный момент активно стали развиваться как китайские, так и корейские аниме, что также добавило в рынок новые сектора.

Аналогичными решениями являются Nekomics (https://nekomics.by/), Superlama (https://superlama.by/), Play-watch(http://play-watch.by/), MyComicBook (https://mycomicbook.by/).

Целевой аудиторией является молодежь.

Бизнес-целью проекта является создание приложения, которое можно будет монетизировать. Текущими задачами для достижения цели являются:

1. Анализ конкурентов и архитектуры их приложений, сервисов.
2. Создание модульной архитектуры приложения.
3. Разработка интерфейса, который будет привлекать пользователей и будет удобным для пользователей всех ролей.
4. Тестирование приложения под нагрузкой, чтобы проверить корректность, устойчивость, безотказность его работы.

В проекте присутствуют различные роли пользователей. Покупатель-гость может просматривать текущие позиции товаров на сайте, использовать фильтры, сравнения товаров, однако не сможет ничего сохранять без личного кабинета. Зарегистрированный покупатель может заполнять данные своего профиля: адрес, ФИО, номер телефона, электронную почту; сохранять сравнения, избранные товары, получать уведомления об отмеченных товарах, на которых появилась скидка, делать заказы, отслеживать их статус, связываться с поддержкой как интернет-магазина, так и с продавцом, в случае проблем с заказом. Зарегистрированный продавец может заполнить свой профиль для покупателей и для администраторов\операторов поддержки, отправлять свой товар на модерацию, предварительно заполнив описание, характеристики, фотографии\видео товара, перед тем как он будет опубликован для покупателей, менять цены на свои опубликованные товары, добавлять скидки, изменять описание, фотографии\видео опубликованного товара, менять статус заказа, связываться с клиентом через чат, или по указанным данным в заказе или профиле клиента, отменять заказы. Администратор может публиковать товары продавцов, блокировать клиентов, продавцов, удалять товары, следить за статистикой продаж, штрафовать продавцов, следить за работоспособностью интернет-магазина. Оператор поддержки может общаться с клиентом или продавцом через чат или по данным, указанным в профилях, передавать решения администратору и продавцу. Проект "Интернет-магазин аниме вещей" построен на современных технологиях: фронтенд - React с использованием хуков, Redux Toolkit для состояния, React Router для навигации, а также библиотеки для форм, уведомлений, иконок и оптимизации. Стили реализованы через CSS Modules, Flexbox, Grid и медиа-запросы для адаптивности. Для взаимодействия с сервером применяется Axios, а аутентификация - через JWT. Данные о корзине, избранных товарах и сравнении хранятся в localStorage. Реализованы поиск, фильтрация, сортировка, рейтинг, отзывы, слайдеры, модальные окна, пагинация и адаптивный дизайн. Архитектура - компонентная, с модульной структурой, кастомными хуками и контекстами. Проект собран с помощью Webpack и Babel, проверяется на качество кода через ESLint. Уделено внимание безопасности (XSS, валидация), производительности (мемоизация, ленивая загрузка) и современному синтаксису JavaScript (ES6+).

### Постановка задачи

### Обзор аналогичных решений

Сайт Nekomics (<https://nekomics.by/>) является одним из самых популярных интернет-магазинов по продаже манги и японской атрибутики. В этом сервисе на главной странице мы сразу видим header в котором реализованы переходы на страницу поиска интересующих нас товаров, логотип сайта, который переносит нас на главную страницу, кнопку в виде сердечка, которая должна направлять нас на страницу с нашими избранными товарами, если мы будем в роли авторизованный покупать. Если мы не авторизованы, то появляется модальное окно, которое предлагает нам войти в личный кабинет или зарегистрироваться. Далее мы видим кнопку корзины, по нажатию на которую, нас переносит на страницу с товарами, которые мы добавили в корзину, в независимости от нашей роли на сайте. Последней в хедере является кнопка профиля. Если мы не авторизованы, то появляется такое же модальное окно, как и при нажатии на кнопку избранных товаров. Ниже у нас идёт nav, в котором расположены категории с выпадающими подменю, в которых расположены подкатегории товаров. Далее мы видим чередующиеся с определенным интервалом 3 информативных слайда. Также ниже присутствуют те же категории, но уже с иллюстрациями конкретных товаров с кнопками которые адресуют нас на страницы каталога с определенными категориями товаров. Ниже мы видим отдельное выделенное место для товаров по определенным тайтлам, карточки с которыми можно проматывать через нажатия на стрелочки по краям страницы. Через кнопки подробнее на этих карточках можно перейти в каталог товаров, соответствующих исключительно выбранному произведению. Еще ниже мы видим такую же систему карточек, но уже с популярными товарами. Но здесь она отличается тем, что мы можем через карточки товара сразу добавить их в корзину или избранное, а также перейти на сам конкретный товар, для изучения его характеристик. Присутствует кнопка “смотреть ещё”, по нажатию на которую нас переносит на каталог популярных товаров. Следующей идет отдел “Новинки”. Здесь все точно также, но на карточках товара еще есть значок “новинка”. Далее мы видим блок новостей, как-либо связанных с тематикой интернет-магазина. Ближе к концу нас встречает блок “Отвечаем на вопросы”, в котором расписаны ответы на наиболее частые вопросы у посетителей сайта. Ещё ниже нас встречает краткое описание интернет-сервиса, а также описание его физической точки. В самом низу всё по стандарту: дублирование категорий, юридические данные, контакты для связи, сертификации, ссылки на политику обработки персональных данных и на пользовательское соглашение.

Каталоги имеют пару фильтров, многие отличаются от каталога к каталогу, но также есть и общие, такие как цена. Снизу каждого каталога какое-то лирическое эссе с внушением что они что-то продают и что мы должны купить именно у них. В каждом каталоге такая “вода” имеет своё содержание и смысл.

Интернет‑магазин Superlama (<https://superlama.by/>) представляет собой пример современного онлайн‑сервиса, в котором акцент сделан на функциональные возможности и удобство взаимодействия с пользователем. В отличие от многих конкурентов, где внимание уделяется прежде всего визуальной эстетике, Superlama демонстрирует богатый набор инструментов, направленных на практическое использование. С первых секунд работы с сайтом пользователь сталкивается с множеством интерактивных элементов. В верхней части страницы расположен счётчик уведомлений, связанный с профилем пользователя. Рядом находится значок профиля, при нажатии на который открывается модальное окно. Это окно не просто предлагает авторизацию или регистрацию, но и объясняет назначение личного кабинета, подчёркивая его ценность для клиента. Такой подход снижает барьер входа для новых пользователей и помогает им понять, зачем нужна регистрация. Поисковая строка реализована с элементами дружелюбного интерфейса: если пользователь нажимает на значок поиска, не введя запрос, система реагирует сообщением «Введите запрос :)». Это создаёт ощущение живого диалога и повышает вовлечённость. Правее поисковой строки находится функционал выбора региона. После выбора система автоматически подбирает магазины, доступные в указанной области. Это решение демонстрирует ориентацию на локализацию и персонализацию сервиса, что особенно важно для интернет‑магазинов, работающих в нескольких регионах. Также в верхней панели присутствует кликабельная ссылка на телефон интернет‑магазина и колл‑центра. Это упрощает связь с поддержкой и повышает доверие клиентов. Значок корзины снабжён счётчиком товаров, а при наведении на пустую корзину появляется подсказка о её состоянии. Такой подход позволяет пользователю быстро оценить текущие действия и состояние заказа. На следующем уровне интерфейса размещён логотип сайта и значок скидок. При нажатии на значок скидок пользователь попадает в Telegram‑канал магазина. Это решение демонстрирует интеграцию с внешними мессенджерами, что расширяет каналы коммуникации и позволяет оперативно информировать клиентов о новых акциях. Ниже расположено меню, которое при наведении раскрывает список франшиз. Пользователь может сразу перейти к товарам, связанным с выбранной франшизой. Такой подход облегчает навигацию и позволяет быстро находить интересующие категории.

Интернет‑магазин Superlama демонстрирует ряд решений, которые могут быть полезны при разработке собственного проекта. Готовые для использования:

1. Счётчик уведомлений и товаров в корзине.
2. Модальные окна для регистрации и авторизации.
3. Подсказки при некорректных действиях пользователя.
4. Кликабельные ссылки для связи с магазином.

Требующие доработки:

1. Интеграция с мессенджерами может быть расширена до полноценного чата поддержки.
2. Региональная персонализация может быть дополнена автоматическим определением местоположения, меню франшиз можно адаптировать под другие категории, например, тематические подборки или бренды.

Интернет-магазин Play-Watch (https://play-watch.by/) представляет собой нишевый сервис, ориентированный на продажу лицензионной аниме-атрибутики, фигурок, сувениров и тематических подарков. С первых секунд пользователь попадает в компактный, но насыщенный интерфейс, где акцент сделан на быстром доступе к ключевым разделам. В верхней части сайта расположен логотип, ведущий на главную страницу, строка поиска, кнопка профиля, корзина и ссылки на социальные сети. При попытке взаимодействия с избранным или профилем без авторизации появляется модальное окно, предлагающее регистрацию или вход в личный кабинет. Это окно сопровождается пояснением, зачем нужен профиль, и какие возможности он открывает.

Корзина снабжена счётчиком товаров, а при наведении на неё отображается всплывающее сообщение о текущем состоянии. Поисковая строка реагирует на пустой запрос дружелюбным сообщением, создавая эффект живого общения. Ниже располагается навигационное меню, разделяющее товары по категориям: фигурки, аксессуары, рюкзаки, одежда, постеры и сувениры. При наведении на категории раскрываются подменю, позволяющие быстро перейти к нужной франшизе или тематике.

На главной странице представлены слайдеры с акциями, новинками и популярными товарами. Карточки товаров снабжены кнопками «в корзину», «в избранное» и «подробнее», позволяющими изучить характеристики и оформить заказ. Некоторые карточки дополнены метками «новинка» или «лицензия», что помогает ориентироваться в ассортименте. Также присутствует кнопка «смотреть ещё», ведущая в расширенный каталог.

Отдельное внимание уделено интеграции с Telegram и VK, через которые магазин продвигает акции, публикует новости и взаимодействует с клиентами. Это создаёт ощущение сообщества и позволяет оперативно информировать покупателей. В нижней части сайта размещены юридические данные, контакты, ссылки на пользовательское соглашение и политику конфиденциальности, что соответствует стандартам e-commerce.

Интернет-магазин Play-Watch демонстрирует ряд решений, которые могут быть полезны при разработке собственного проекта. Готовые для использования:

1. Счётчик товаров в корзине и всплывающие подсказки.
2. Модальные окна для регистрации и авторизации.
3. Быстрые действия на карточках товара.
4. Интеграция с социальными сетями для продвижения и поддержки. Требующие доработки:
5. Расширение фильтров по франшизам, брендам и редкости.
6. Внедрение системы предзаказов и уведомлений о поступлениях.
7. Развитие поддержки через чат или бот.
8. Синхронизация онлайн и офлайн-остатков, если предполагается физическая точка.

Интернет-магазин MyComicBook (https://mycomicbook.by/) - это специализированная платформа по продаже комиксов, манги, ранобэ, артбуков и тематических книг. Сайт ориентирован на поклонников графических историй и предлагает широкий ассортимент продукции от ведущих издательств. На главной странице пользователь сразу видит верхнюю панель, включающую логотип, строку поиска, корзину, кнопку профиля и ссылки на регистрацию. При попытке взаимодействия с личным кабинетом или корзиной без авторизации появляется модальное окно, предлагающее войти или зарегистрироваться.

Поисковая строка позволяет искать товары по названию, автору или серии. Корзина отображает количество добавленных товаров, а при наведении на неё появляется сообщение о текущем состоянии. Навигационное меню делит каталог на основные категории: манга, комиксы, артбуки, ранобэ, книги, мерч и настольные игры. При наведении на разделы раскрываются подкатегории, позволяющие быстро перейти к нужному типу продукции.

На главной странице размещены блоки «Новинки», «Под заказ», «Предзаказ» и «Скидки». Карточки товаров содержат изображение, цену, статус доступности и кнопки «в корзину» и «подробнее». Некоторые карточки дополнены метками «эксклюзив» или «лимитированное издание», что подчёркивает уникальность предложения. Также присутствует кнопка «где мой заказ», позволяющая отслеживать статус доставки.

Сайт активно публикует новости, связанные с выходом новых томов, акциями и мероприятиями. В нижней части страницы размещены юридические сведения, контакты, ссылки на пользовательское соглашение, политику конфиденциальности и условия возврата. Это создаёт ощущение надёжности и прозрачности.

Интернет-магазин MyComicBook демонстрирует ряд решений, которые могут быть полезны при разработке собственного проекта. Готовые для использования:

1. Структурированный каталог с подкатегориями.
2. Система предзаказов и статусов «под заказ».
3. Модальные окна для авторизации и отслеживания заказов.
4. Метки «эксклюзив» и «лимит» на карточках товара. Требующие доработки:
5. Расширение фильтров по авторам, издательствам и жанрам.
6. Внедрение персональных рекомендаций на основе истории покупок.
7. Интеграция с соцсетями и мессенджерами для поддержки и продвижения.
8. Возможность формирования коллекций и списков желаемого.

### 1.2 Техническое задание

Целью проекта является создать современный интернет‑магазин для продажи аниме‑атрибутики, сувениров и тематических товаров. Сайт должен обеспечивать удобный интерфейс, высокую производительность, безопасность и адаптивность под различные устройства.

Задачами проекта являются:

1. Разработать архитектуру сайта с разделением на клиентскую и серверную части.
2. Реализовать систему регистрации и авторизации пользователей.
3. Обеспечить функционал корзины, избранного и сравнения товаров.
4. Реализовать поиск и фильтрацию товаров по различным критериям.
5. Обеспечить возможность оформления заказов и просмотра истории покупок.
6. Создать адаптивный дизайн для корректного отображения на ПК, планшетах и смартфонах.
7. Обеспечить защиту данных пользователей и корректную работу под нагрузкой.

Требованиями к функционалу являются:

1. Регистрация и вход пользователей с использованием JWT‑токенов.
2. Личный кабинет с историей заказов и настройками профиля.
3. Каталог товаров с фильтрацией по цене, категории, бренду, франшизе.
4. Возможность добавления товаров в корзину, избранное и список сравнения.
5. Оформление заказа с выбором способа доставки и оплаты.
6. Система отзывов и рейтингов для товаров.
7. Слайдеры и карусели для отображения акций и новинок.
8. Модальные окна для детальной информации о товаре.
9. Пагинация каталога.
10. Интеграция с социальными сетями для продвижения и обратной связи.

Требованиями к интерфейсу являются:

1. Простая и интуитивная навигация.
2. Единый стиль оформления, соответствующий тематике аниме.
3. Адаптивный дизайн (desktop, tablet, mobile).
4. Использование иконок и визуальных меток («новинка», «акция», «лицензия»).
5. Уведомления о действиях пользователя (добавление в корзину, успешная регистрация и т.д.).

Требованиями к архитектуре являются:

1. Компонентная архитектура на основе React.
2. Управление состоянием через Redux Toolkit.
3. Разделение кода на модули: components, pages, store, services, utils.
4. Использование хуков React и кастомных хуков для повторного использования логики.
5. Асинхронные операции через Axios и async/await.
6. Сохранение состояния корзины и избранного в localStorage.

Требованиямик безопасности являются:

1. Аутентификация через JWT‑токены.
2. Валидация входных данных на клиенте и сервере.
3. Защита от XSS и CSRF‑атак.
4. Шифрование паролей и персональных данных.

Нефункциональными требованиями являются:

1. Производительность: время отклика не более 2 секунд при стандартной нагрузке.
2. Масштабируемость: возможность добавления новых категорий и модулей без изменения основной архитектуры.
3. Надёжность: корректная работа при одновременном использовании до 1000 пользователей.
4. Кроссбраузерность: поддержка последних версий Chrome, Firefox, Edge, Safari.

Инструментами разработки являются:

1. Webpack и Babel для сборки и транспиляции.
2. ESLint для контроля качества кода.
3. Create React App для инициализации проекта.
4. Git для контроля версий.

Ожидаемым результатом является:

Готовый интернет‑магазин аниме вещей с современным интерфейсом, безопасностью, высокой производительностью и возможностью дальнейшего масштабирования.

### 1.3 Выбор средств реализации программного продукта

На начальном этапе создания веб-сайта разрабатывается его визуальное оформление. Для этого целесообразно использовать графический онлайн-редактор Figma. В Figma возможно проектирование элементов интерфейса, создание интерактивного прототипа веб-сайта или приложения, разработка иллюстраций и векторной графики. Визуальное представление данного проекта, разработанное в Figma, представлено в Приложении А.

Для формирования структуры веб-сайта был выбран язык гипертекстовой разметки HTML. Данный язык представляет собой набор инструкций, посредством которых браузеры отображают на экране различные документы и веб-страницы. HTML формирует основу внутренней организации веб-сайта, его фундаментальный каркас. В проекте используются семантические элементы HTML5 для создания логической структуры страниц, включая заголовки, навигацию, основное содержимое, формы и футер.

Для определения визуального представления страниц веб-сайта применяется язык CSS, который задает визуальные характеристики элементов и контролирует их расположение на странице. CSS предлагает широкий спектр инструментов, включая каскадное применение стилей, адаптивный дизайн, создание анимационных эффектов и трансформаций, использование сеток (Grid), гибких блоков (Flexbox), псевдоклассов и других возможностей. В проекте также используется препроцессор SCSS для более удобной и структурированной работы со стилями, включая переменные, миксины и вложенность.

Для обеспечения динамического поведения веб-сайта будет использован язык JavaScript – многопарадигменный язык программирования. В проекте применяется современный синтаксис ES6+, включая стрелочные функции, деструктуризацию, модули, промисы и async/await для асинхронных операций. JavaScript используется для реализации интерактивных элементов интерфейса, управления состоянием приложения, работы с API, обработки событий и взаимодействия с DOM.

Разработка структуры веб-сайта осуществляется в среде Visual Studio Code, предлагающей функциональность отладки, интеграцию с системой контроля версий Git и выделение синтаксиса. Платформа отличается гибкостью настройки, позволяя применять персональные темы оформления, назначать сочетания клавиш и модифицировать файлы конфигурации для адаптации к индивидуальным потребностям разработчика. VS Code также предоставляет расширения для поддержки HTML, CSS, JavaScript, SCSS и других технологий, используемых в проекте.

### 1.4 Выводы по разделу

На основе анализа аналогичных интернет-магазинов выявлены ключевые функциональные и дизайнерские элементы успешных решений: модальные окна авторизации, счётчики товаров в корзине, быстрые действия на карточках товаров, расширенные фильтры по категориям и брендам, слайдеры для акций и новинок, интеграция с социальными сетями. Сформулированы цели и задачи проекта - разработка современного, безопасного и масштабируемого интернет-магазина аниме-атрибутики с адаптивным интерфейсом, поддержкой личного кабинета, корзины, избранного, каталога с фильтрацией и оформлением заказов. Выбран стек технологий: React с Redux Toolkit, HTML5, CSS/SCSS, JavaScript (ES6+), Figma для прототипирования, VSCode для разработки. Проект соответствует требованиям безопасности, производительности, кроссбраузерности и готов к дальнейшему масштабированию.

### Проектирование страниц веб-сайта

### 2.1 Общая структура

Проект веб-сайт интернет-магазина имеет следующую структуру, представленную на рисунках 2.1 – 2.4

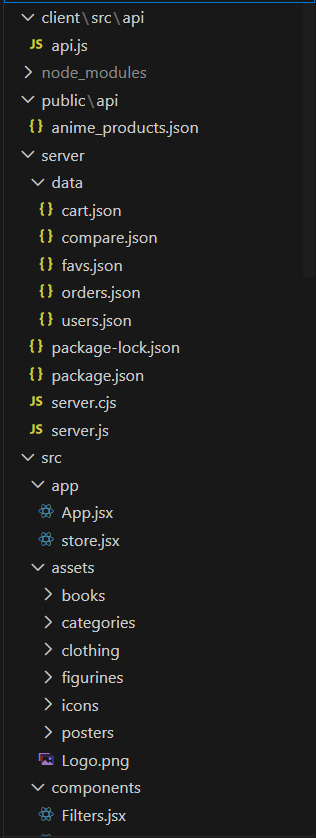


Рисунок 2.1

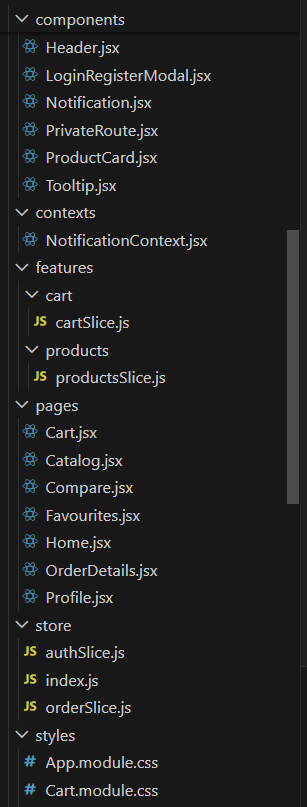


Рисунок 2.2

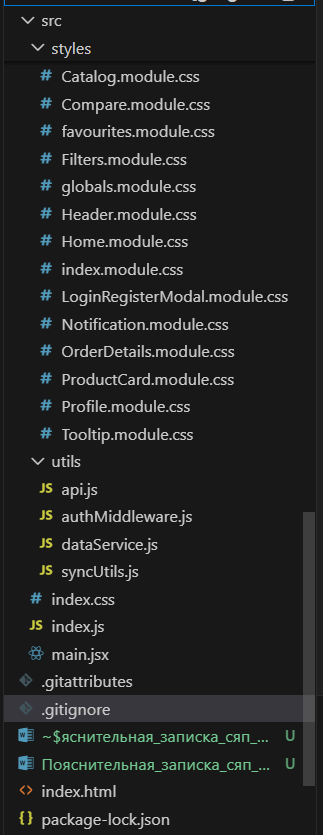


Рисунок 2.3

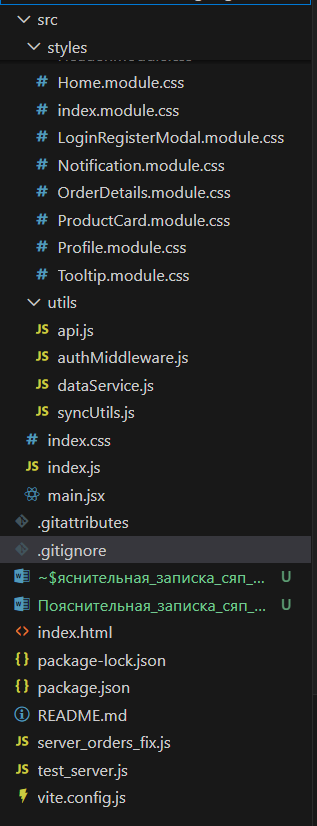


Рисунок 2.4

### 2.2 Выбор способа верстки

В проекте использовалась адаптивная верстка на основе HTML5, Flexbox, CSS Grid и медиа-запросов с применением относительных единиц измерения для обеспечения кросс-платформенной совместимости и плавного масштабирования интерфейса.

### 2.3 Выбор стилевого оформления

Выбрано стилевое оформление с использованием CSS-модулей, SCSS, CSS-переменных, Flexbox и Grid для гибкой, адаптивной и визуально привлекательной структуры с единым цветовым акцентом, тематическими шрифтами и плавными анимациями.

### 2.4 Выбор шрифтового оформления

Шрифтовое оформление основано на использовании Google Fonts: основной шрифт - Inter для обеспечения читаемости и современного вида, заголовки и акценты - Montserrat для визуальной выразительности. Применены CSS-переменные для согласованного управления размерами и насыщенностью, с градацией от 100 до 900, обеспечивающей иерархию текста. Размеры: заголовки H1–H6 - от 16 до 48px, основной текст - 14–16px, подзаголовки и подписи - 12–14px. Насыщенность: 400 (regular), 500 (medium), 600 (semibold), 700 (bold) в зависимости от значимости. Адаптивность достигается через использование rem и em, а также медиа-запросов для мобильных устройств. Улучшена читаемость за счёт line-height (1.4–1.6) и letter-spacing (-0.02em). Цветовая палитра: тёмные оттенки (#000, #333, #666) для основного контента, светлые (#999, #ccc) - для второстепенной информации. Для аниме-тематики добавлены акцентные стили с эффектами, используемые для названий товаров и акционных элементов.

### 2.5 Разработка логотипа

Логотип разработан в Figma как векторный элемент с аниме-стилистикой: сочетание иероглифов, катаны и манги с градиентом от фиолетового к розовому, выполненный в шрифте Montserrat Semi-Bold. Используется как элемент навигации на главную страницу.



Рисунок 2.5 – Логотип сайта

### 2.6 Разработка пользовательских элементов

Пользовательские элементы включают кнопки (primary, secondary, icon) с hover-эффектами и анимацией; карточки товаров с изображением, ценой, рейтингом, кнопками "в корзину", "в избранное", "сравнить"; формы (логин, регистрация, контактная) с валидацией и плейсхолдерами; навигационное меню с выпадающими списками; слайдеры (акции, новинки) с автопрокруткой и навигацией; фильтры (цена, категория, бренд) с чекбоксами и слайдерами; табы (описание, отзывы, характеристики); модальные окна (просмотр товара, корзина, профиль) с затемнением фона; прогресс-бары; тултипы; уведомления (success, error, warning) с автозакрытием; иконки (корзина, профиль, избранное) с изменением цвета при активности; пагинацию с цифровой навигацией; счетчики (товаров в корзине, уведомлений); инпуты с иконками и валидацией; теги ("новинка", "акция", "топ") с цветовой дифференциацией; рейтинговые звезды (интерактивные); хлебные крошки; спиннеры загрузки; якорные ссылки; тени и градиенты для акцентов; SVG-иконки для векторной отрисовки; адаптивные элементы для мобильных устройств. Все компоненты реализованы как переиспользуемые React-компоненты с CSS-модулями и SCSS.

### 2.7 Разработка спецэффектов

Спецэффекты включают hover-анимации, transition-эффекты, fade-in/out, scale- и rotate-анимации, pulse-, slide-, bounce-, loader-, parallax-, glow-, flip-, shake-, ripple-, blur-, zoom-, marquee-, text-shadow-, gradient-, opacity-, box-shadow-, border-animation и scroll-triggered-анимации. Реализованы с помощью CSS-анимаций, keyframes, transitions, transform, filter и JavaScript-библиотек (Framer Motion, AOS).

### 2.8 Выводы по разделу

В разделе "Проектирование страниц веб-сайта" реализован комплексный подход к созданию современного, адаптивного и визуально выразительного интерфейса. Адаптивная верстка на основе HTML5, Flexbox, CSS Grid и медиа-запросов обеспечивает стабильную работу сайта на всех устройствах. Использование SCSS, CSS-переменных и модульных стилей позволяет поддерживать единый визуальный стиль и упрощает сопровождение кода. Выбор шрифтов Inter и Montserrat с градацией начертаний и адаптивными размерами гарантирует высокую читаемость и иерархию текста. Логотип в аниме-стилистике, разработанный как векторный элемент, усиливает тематическую идентичность проекта. Все пользовательские элементы - от кнопок и карточек до форм, фильтров и модальных окон - реализованы как переиспользуемые компоненты с соблюдением принципов доступности и удобства. Спецэффекты, включая анимации, переходы и интерактивные эффекты, повышают вовлечённость пользователя. В совокупности данные решения обеспечивают функциональный, эстетически привлекательный и полностью адаптивный интерфейс, соответствующий требованиям аниме-тематического онлайн-магазина.

### 3 Реализация структуры веб-сайта

### 3.1 Структура HTML-документа

Основой проекта является index.html с семантической разметкой: <!DOCTYPE html>, <html lang="ru">, <head> с мета-тегами (charset, viewport, description, keywords, title), подключением шрифтов Google Fonts (Inter, Montserrat), иконки favicon, а также подключением CSS-файлов. В теле страницы находится <div id="root"> - корневой элемент для React-приложения, скрипты type="module" и nomodule для совместимости, реализованная навигация через BrowserRouter, Routes и Route с инкапсуляцией страниц: Главная, Каталог, Корзина, Избранное, Профиль, Оформление заказа, Подробности товара. Добавлены: подключение стилей, иконок (Font Awesome, React Icons), шрифтов, глобальные стили сброса (reset.css/normalize.css). Структура включает <header> с логотипом, поиском, навигацией, кнопками корзины и профиля; <main> с динамическим контентом; <footer> с юридической информацией, ссылками и контактами. Для SEO и доступности используются: aria-label, role, tabindex, семантические теги (<nav>, <article>, <section>, <aside>), meta-теги Open Graph и Twitter Card. Подключены: manifest.json для PWA, sitemap.xml, robots.txt, preload для критических ресурсов.

### 3.2 Структура JSX-файлов

Компоненты организованы по директориям: components (общие элементы: кнопки, карточки, формы, слайдеры), pages (страницы: Main, Catalog, ProductDetail, Cart, Profile), store (логика состояния через Redux), services (API-вызовы), utils (вспомогательные функции). В App.jsx настроена маршрутизация с react-router-dom, каждый маршрут связан с соответствующей страницей. Страницы используют хуки: useEffect, useState, useSelector, useDispatch для управления состоянием и побочными эффектами. Компоненты карточек товаров содержат: изображение (<img> с alt), название, цену, рейтинг, кнопки действий. Формы (Login, Register) включают валидацию через React Hook Form и интеграцию с API. Управление состоянием корзины, избранного и сравнения реализовано через Redux-хуки. Каталоги поддерживают пагинацию, фильтры (через useSearchParams), сортировку и lazy loading. Страница профиля интегрирована с JWT-аутентификацией, отображает историю заказов, настройки, форму редактирования. Модальные окна используют порталы (createPortal) и контексты (Context API) для изоляции стилей и управления. Все компоненты стилизуются через CSS-модули и SCSS, используют PropTypes или TypeScript для проверки пропсов. Асинхронные операции реализованы с async/await, try/catch, loading/error-состояниями. Реализованы: ErrorBoundary, LazyLoading для изображений, debounce для поиска, memo/useMemo/useCallback для оптимизации.

### 3.3 Добавление таблицы стилей CSS

Стили реализованы через модульный подход: CSS-модули, SCSS, переменные CSS, Flexbox, Grid. В index.scss заданы глобальные стили: сброс (reset), шрифты, цвета, размеры, тени. В variables.scss определены: $primary-color, $secondary-color, $font-family, $breakpoints для адаптивности. Каждый компонент имеет собственный SCSS-файл (например, Button.module.scss), подключаемый через import styles from './Component.module.scss'. Используются: @mixin для повторяющихся блоков (например, button-style, card-layout), @include для включения миксинов, @media для адаптивности. Анимации реализованы через transition, transform, keyframes (hover-эффекты, плавные появления, спиннеры). Адаптивность достигается через min-width, max-width, rem/em, vw/vh. В theme.scss - темы (светлая/тёмная), переключаемые через CSS-переменные. В animations.scss - общие анимации: fadeIn, slideIn, pulse. Применяются: z-index, box-shadow, border-radius, gradient, filter (blur для модальных окон). Иконки - SVG-спрайты или react-icons. Для доступности: focus, outline, high-contrast. В процессе сборки - минификация, объединение, оптимизация стилей через Webpack-плагины.

### 3.4 Выводы по разделу

Структура веб-сайта построена на современном стеке: семантический HTML5, модульные JSX-компоненты, адаптивный и масштабируемый CSS/SCSS. HTML-документ обеспечивает корректную загрузку, доступность, SEO и совместимость. JSX-файлы реализованы по принципам модульности, используя хуки, контексты, маршрутизацию и управление состоянием. Стили применяются через модульный подход, переменные, миксины, адаптивные брейкпоинты и анимации. В совокупности это гарантирует высокую производительность, поддерживаемость, кросс-браузерность и удобство дальнейшего развития проекта.

### Тестирование веб-сайта

### 4.1 Адаптивный дизайн веб-сайта

Адаптивный дизайн реализован с использованием медиа-запросов, Flexbox и Grid Layout. Применены относительные единицы измерения (%, rem, em), а также свойства max-width и min-width для обеспечения корректного отображения на различных устройствах. Интерфейс адаптируется под три группы экранов: мобильные (до 768px), планшетные (768px–1024px) и десктопные (от 1024px). Навигация при уменьшении экрана сворачивается в бургер-меню, карточки товаров перестраиваются в одну колонку, шрифты и отступы масштабируются, элементы интерфейса перераспределяются для удобства работы с тач-интерфейсом. Реализован подход mobile-first: базовые стили разрабатываются для мобильных устройств, затем добавляются стили для больших экранов. Адаптивность охватывает все ключевые компоненты: шапка сайта, навигация, каталог товаров, формы, карточки, слайдеры, футер. В некоторых элементах учтена смена ориентации экрана через медиа-запросы.

### 4.2 Кроссбраузерность веб-сайта

Кроссбраузерность сайта обеспечивается применением современных, широко поддерживаемых технологий: HTML5, CSS3 (Flexbox, Grid, анимации), ES6+ JavaScript. Используются префиксы для старых версий браузеров (-webkit, -moz, -ms), полифилы для поддержки новых функций (Promise, fetch). В CSS применяются fallback’и для градиентов, теней, флексов. Проверка проводилась в последних версиях Chrome, Firefox, Safari, Edge, частично - в Opera и мобильных браузерах. Применяются современные фреймворки (React, Redux), обеспечивающие совместимость. Для стилей используется PostCSS с Autoprefixer. Обеспечено корректное отображение верстки, шрифтов, иконок, форм, анимаций, взаимодействие с DOM, работа API-запросов, отображение SVG и изображений. Экспериментальные технологии без полифилов не используются.

### 4.3 Выводы по разделу

В разделе "Тестирование веб-сайта" подтверждена эффективность адаптивного дизайна и кроссбраузерной совместимости. Адаптация интерфейса под разные устройства (mobile, tablet, desktop) реализована через медиа-запросы, Flexbox/Grid, относительные единицы и подход mobile-first. Кроссбраузерность обеспечена использованием современных стандартов, префиксов, полифилов и проверкой в актуальных браузерах. Это гарантирует стабильную работу и визуальную целостность сайта вне зависимости от устройства или браузера пользователя.

### 5.1 Руководство для пользователя

При открытии сайта вы попадаете на главную страницу. Клик по логотипу возвращает на главную. В шапке: строка поиска (ввод - результаты, кнопка "🔍" - поиск), иконки корзины, избранного, сравнения, профиля.  
Клик по категории (фигурки, книги, одежда, плакаты) - переход в соответствующий каталог.  
Поиск по названию товара или персонажа - отображение результатов с возможностью перехода по клику.  
В каталоге: фильтры (название, категория, бренд, цена), карточки товаров с кнопками: "Добавить в корзину", "Избранное" (сердце), "Сравнение" (шкала), "Подробнее".  
Кнопка "Добавить в корзину" - добавление товара; при неавторизованном пользователе - окно входа/регистрации.  
Иконка корзины в шапке - переход в корзину; там можно изменить количество, удалить товар, оформить заказ (форма: имя, телефон, адрес, email, способ оплаты).  
Иконка сердца - переход в избранное; там можно добавить в корзину или удалить.  
Иконка сравнения - переход к сравнению товаров; можно сравнивать характеристики, удалять, очищать.  
Иконка профиля - переход в личный кабинет; при неавторизации - вход/регистрация. В профиле: редактирование данных, история заказов, выход.  
Клик по заказу в истории - детали заказа: номер, дата, статус, способ оплаты, список товаров, сумма.  
Кнопка "Подробнее" в карточке товара - детальная страница товара с описанием, характеристиками, рейтингом, возможностью добавления в корзину.  
Кнопка "Сбросить фильтры" - сброс всех фильтров в каталоге.  
В профиле - кнопка "Выйти" для выхода из аккаунта.

### 5.2 Выводы по разделу

Веб-сайт удобен и интуитивно понятен: все основные функции - от навигации до оформления заказа - реализованы логично и доступно.

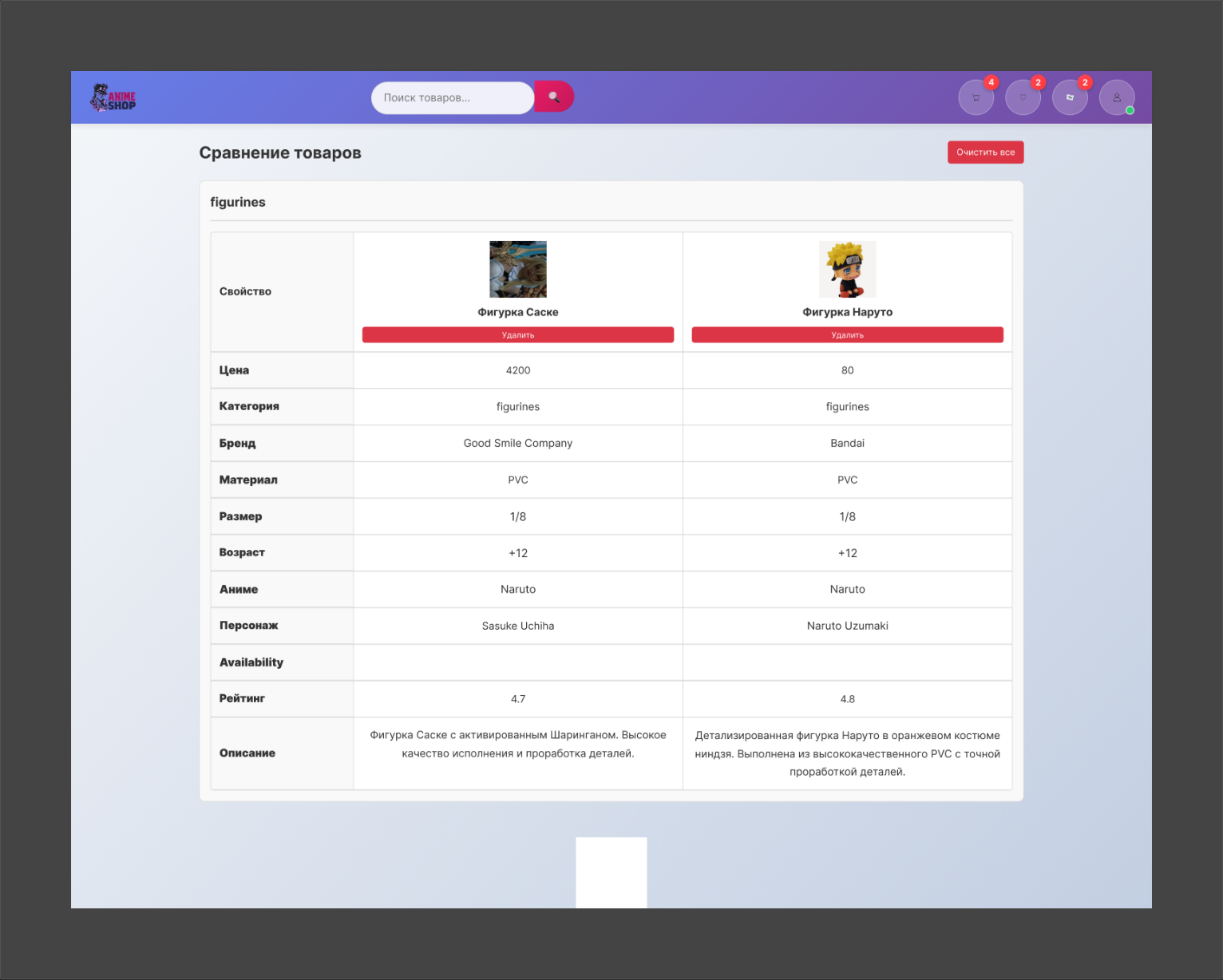
### Заключение

В заключение, цель проекта - создание интернет-магазина аниме-атрибутики с функционалом корзины, избранного, личного кабинета и адаптивным интерфейсом. Были выполнены все этапы разработки: анализ аналогов, выбор стиля, создание логотипа, дизайн интерфейса, внедрение визуальных эффектов, обеспечение адаптивности и кроссбраузерной совместимости. Результат - полностью работоспособный сайт, соответствующий целевому назначению.

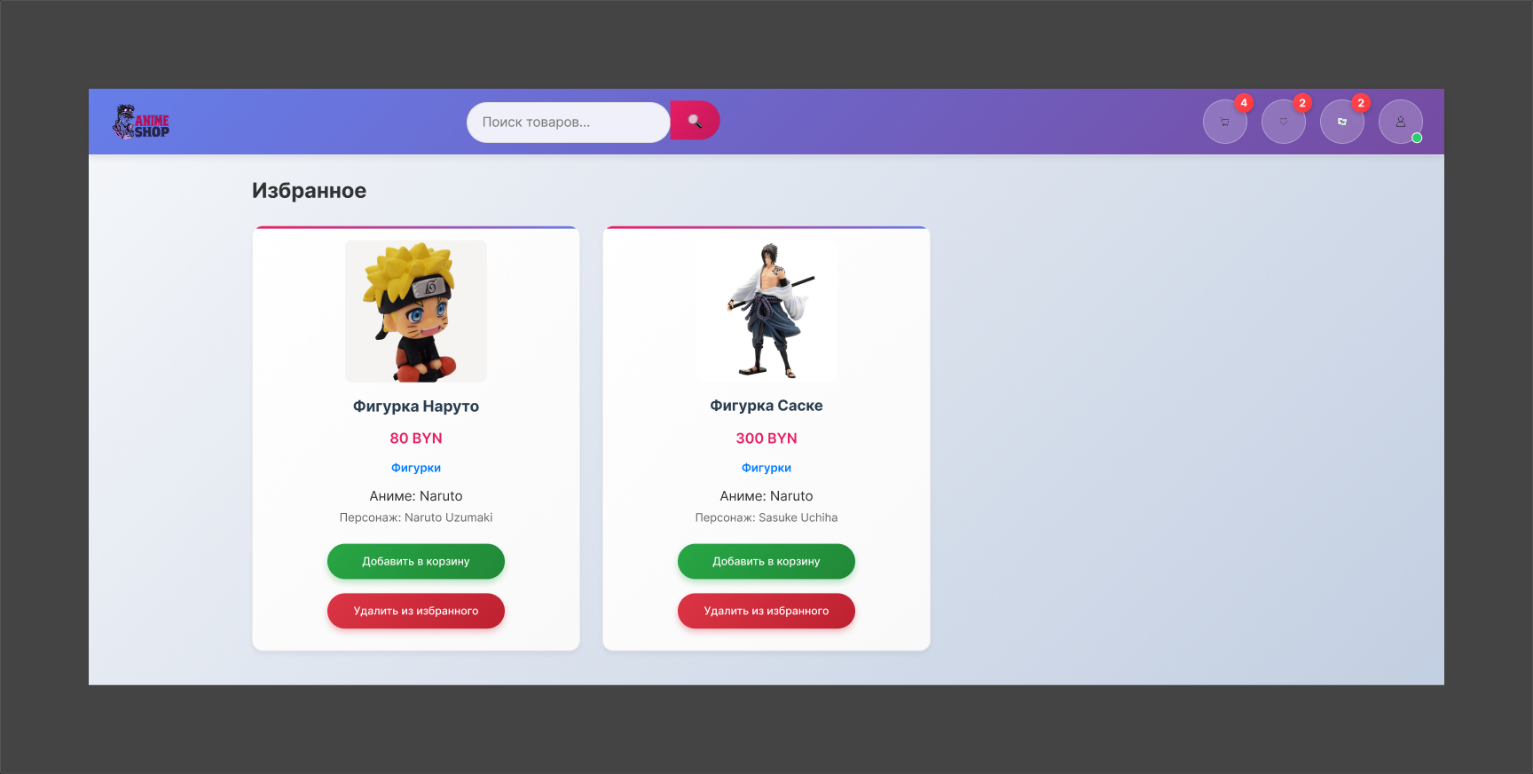
### Список использованных литературных источников

1. Справочник по HTML / Html book – режим доступа: [http://htmlbook.ru](http://htmlbook.ru/) – Дата доступа: 02.03.2025.
2. Адаптивная вёрстка, урок второй [Электронный ресурс] – режим доступа: <https://html5book.ru/adaptivnoe-video/> – Дата доступа: 10.03.2025.
3. Sass / Основы Sass – режим доступа: <https://sass-scss.ru/guide/> – Дата доступа: 29.03.2025.
4. Кобайло, А. С. Введение в XML: учеб.-метод. пособие для студентов специальности «Информационные системы и технологии» / А. С. Кобайло, Н. А. Жиляк. – Минск: БГТУ, 2011. – Дата доступа: 07.04.2025.
5. Дакетт, Д. HTML и CSS. Разработка и создание веб-сайтов / Джон Дакетт; [пер. с англ. М. А. Райтмана]. – Москва: Эксмо, 2020. – Дата доступа: 16.04.2025.
6. React Documentation [Электронный ресурс] – режим доступа: <https://react.dev/> – Дата доступа: 15.03.2025.
7. Redux Toolkit Guide [Электронный ресурс] – режим доступа: <https://redux-toolkit.js.org/> – Дата доступа: 18.03.2025.
8. CSS-Tricks: Flexbox и Grid [Электронный ресурс] – режим доступа: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/> – Дата доступа: 20.03.2025.
9. MDN Web Docs: JavaScript [Электронный ресурс] – режим доступа: <https://developer.mozilla.org/ru/docs/Web/JavaScript> – Дата доступа: 25.03.2025.
10. Figma Community: UI Design [Электронный ресурс] – режим доступа: <https://www.figma.com/community> – Дата доступа: 05.04.2025.
11. GitHub: Web Development Resources [Электронный ресурс] – режим доступа: <https://github.com/topics/web-development> – Дата доступа: 10.04.2025.
12. Webpack Documentation [Электронный ресурс] – режим доступа: <https://webpack.js.org/> – Дата доступа: 12.04.2025.
13. ESLint: Code Quality Tool [Электронный ресурс] – режим доступа: <https://eslint.org/> – Дата доступа: 14.04.2025.
14. Google Fonts [Электронный ресурс] – режим доступа: <https://fonts.google.com/> – Дата доступа: 16.04.2025.
15. W3C Standards [Электронный ресурс] – режим доступа: <https://www.w3.org/standards/> – Дата доступа: 18.04.2025.

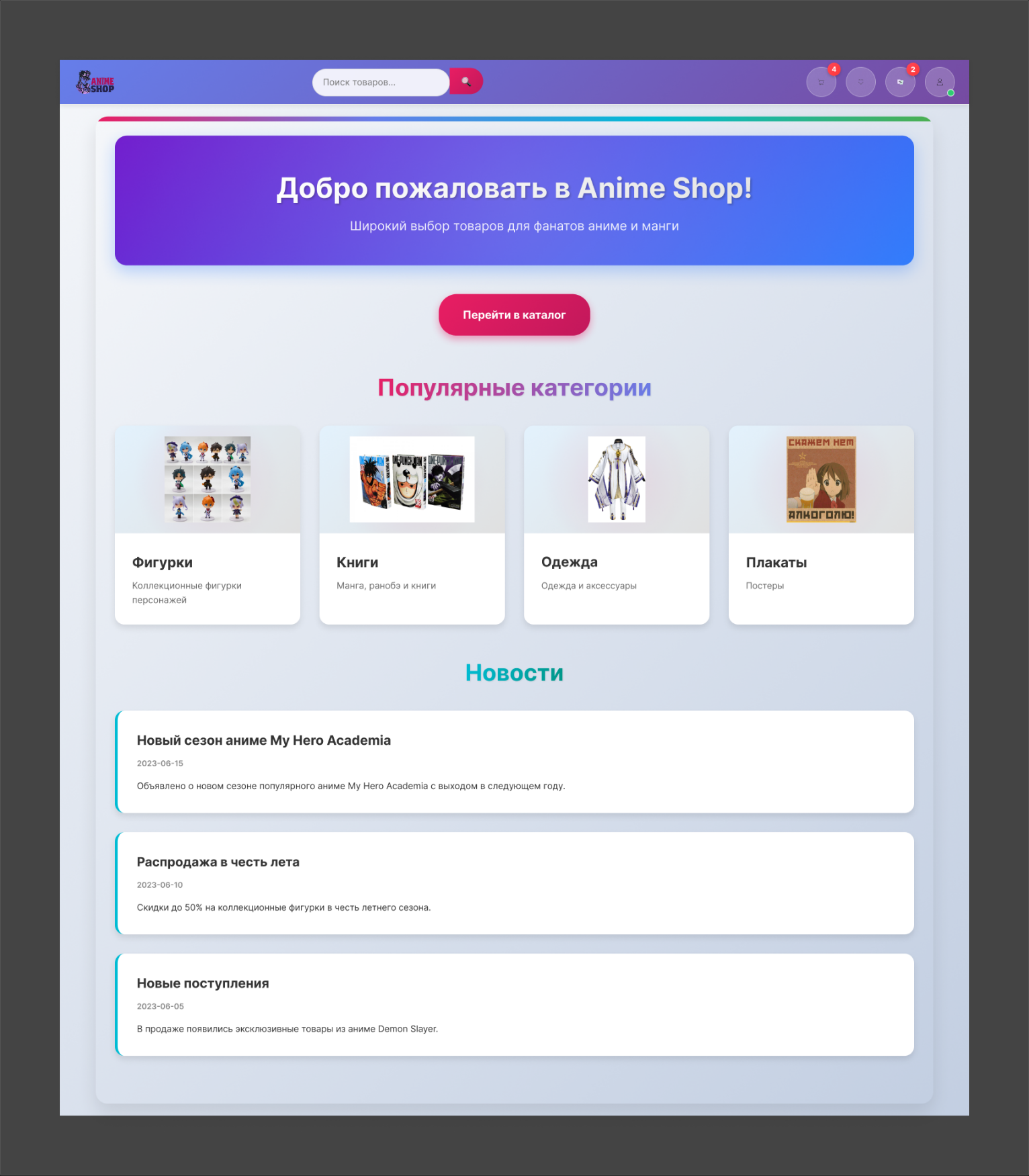
### ПРИЛОЖЕНИЕ А. Макеты веб-страниц



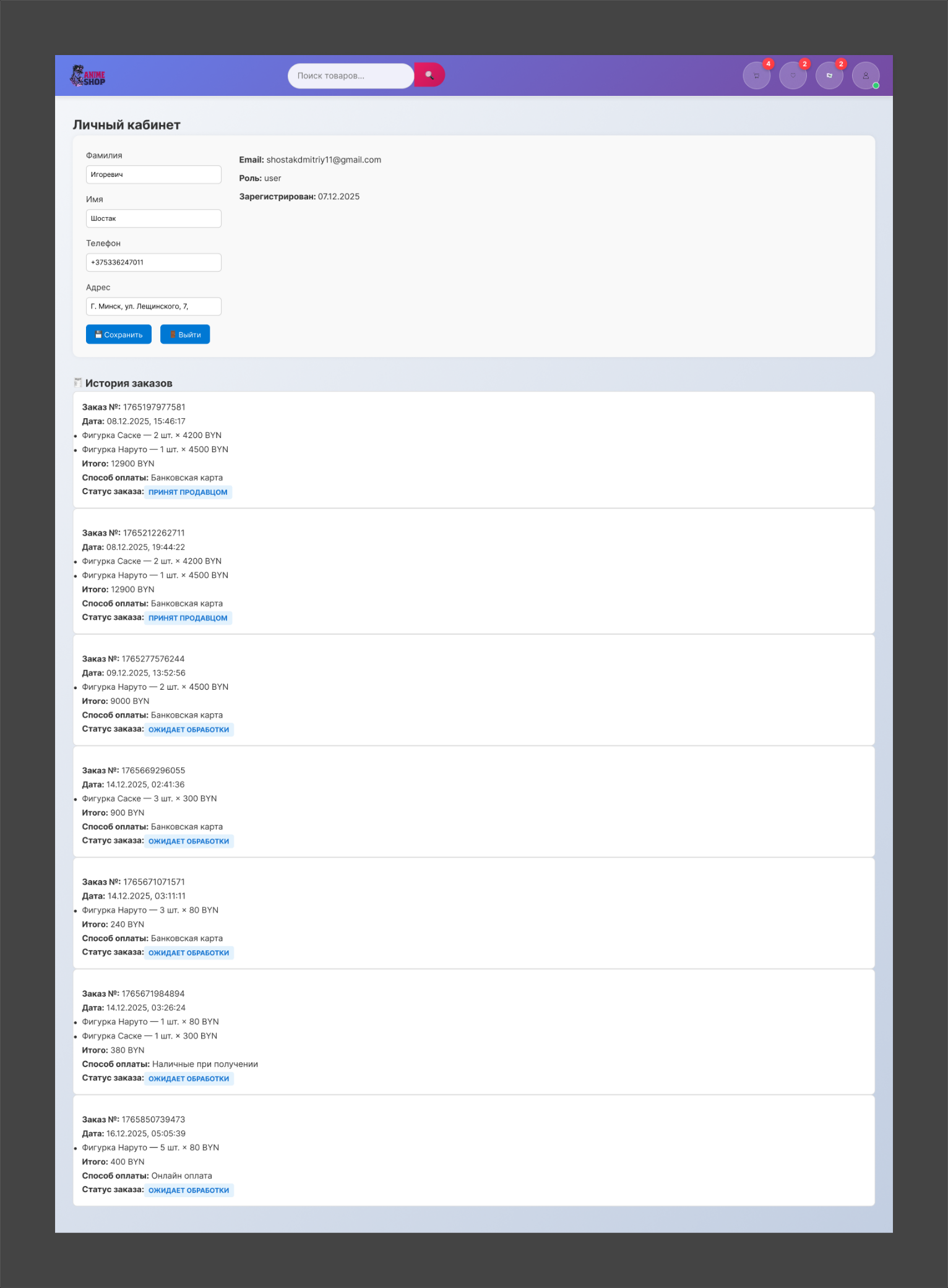
Изображение 1 – макет compare.jsx

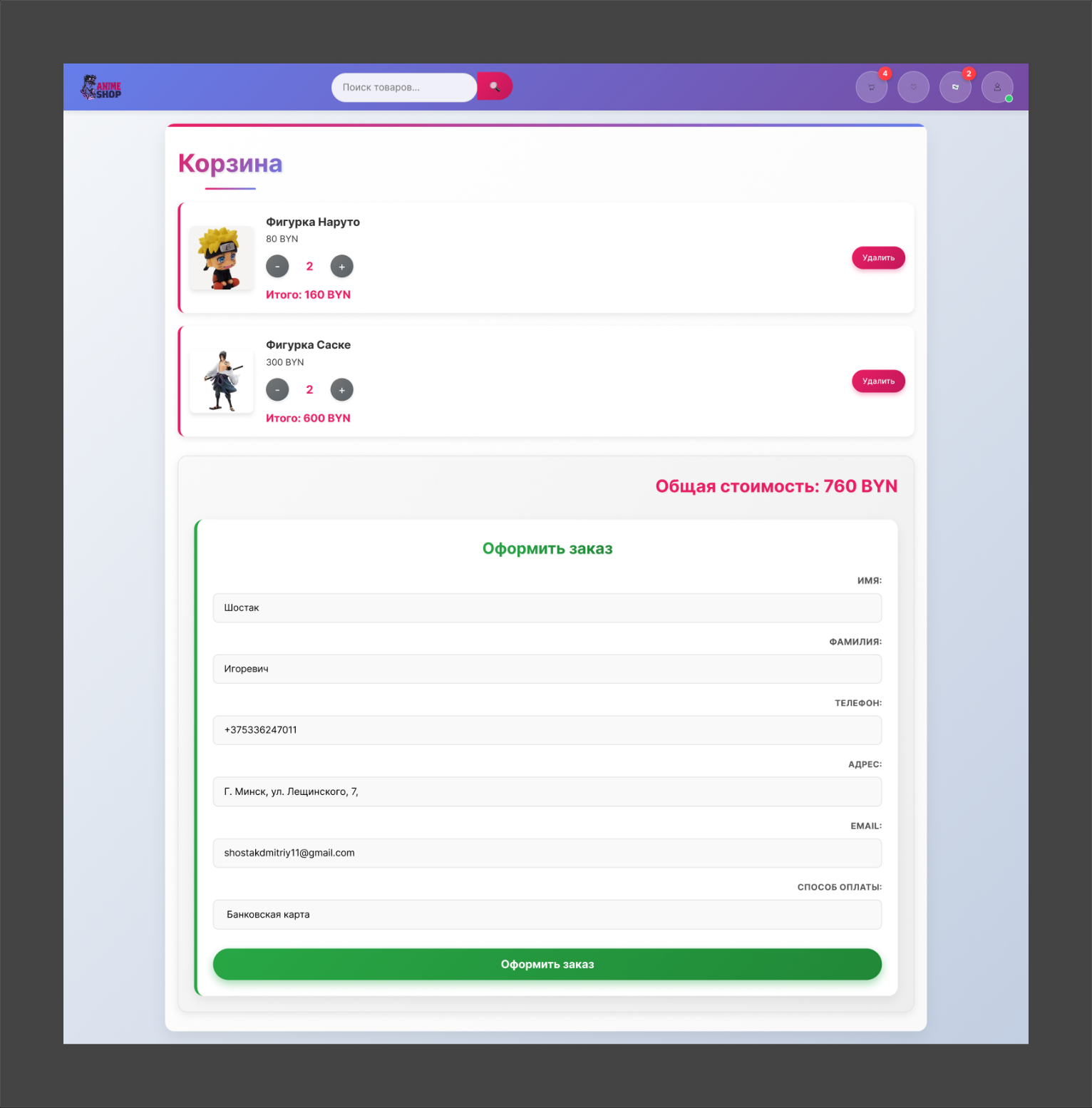


Изображение 2 – макет favourites.jsx



Изображение 3 – макет main.jsx



Изобрадение 4 -макет profile.jsx

Изображение 5 – макет cart.jsx



Изображение 6 – макет catalog.jsx

### ПРИЛОЖЕНИЕ Б. Листинг Html документов

<!doctype html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <link rel="icon" type="image/svg+xml" href="/vite.svg" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Anime Shop</title>

  </head>

  <body>

    <div id="root"></div>

    <script type="module" src="/src/main.jsx"></script>

  </body>

</html>

Листинг 1 – index.html

### ПРИЛОЖЕНИЕ В. Листинг js/jsx документов

const API\_ROOT = 'http://localhost:3001/api';

// Функция для добавления токена авторизации к запросу

function getAuthHeaders() {

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

  const headers = { 'Content-Type': 'application/json' };

  if (token) {

    headers['Authorization'] = `Bearer ${token}`;

  }

  return headers;

}

async function apiFetch(path, opts = {}) {

  const headers = opts.headers || getAuthHeaders();

  const res = await fetch(`${API\_ROOT}${path}`, {

    headers,

    ...opts

  });

  const text = await res.text();

  try {

    const json = text ? JSON.parse(text) : {};

    if (!res.ok) throw { status: res.status, body: json };

    return json;

  } catch (err) {

    if (err && err.status) throw err;

    throw { status: res.status, message: 'Invalid JSON response' };

  }

}

export async function register(data) {

  return apiFetch('/register', { method: 'POST', body: JSON.stringify(data) });

}

export async function login(data) {

  return apiFetch('/login', { method: 'POST', body: JSON.stringify(data) });

}

export async function getUser(id) {

  return apiFetch(`/users/${id}`, { method: 'GET' });

}

export async function updateUser(id, data) {

  return apiFetch(`/users/${id}`, { method: 'PUT', body: JSON.stringify(data) });

}

// Orders

export async function createOrder(order) {

  return apiFetch('/orders', { method: 'POST', body: JSON.stringify(order) });

}

export async function getOrdersByUser(userId) {

  return apiFetch(`/orders/${userId}`, { method: 'GET' });

}

// Cart

export async function getCart(userId) {

  return apiFetch(`/cart/${userId}`, { method: 'GET' });

}

export async function updateCart(userId, items) {

  return apiFetch(`/cart/${userId}`, { method: 'PUT', body: JSON.stringify({ items }) });

}

// Favs

export async function getFavs(userId) {

  return apiFetch(`/favs/${userId}`, { method: 'GET' });

}

export async function updateFavs(userId, items) {

  return apiFetch(`/favs/${userId}`, { method: 'PUT', body: JSON.stringify({ items }) });

}

// Compare

export async function getCompare(userId) {

  return apiFetch(`/compare/${userId}`, { method: 'GET' });

}

export async function updateCompare(userId, items) {

  return apiFetch(`/compare/${userId}`, { method: 'PUT', body: JSON.stringify({ items }) });

}

Листинг 1 – api.js

const express = require('express');

const fs = require('fs');

const path = require('path');

const cors = require('cors');

const { v4: uuidv4 } = require('uuid');

const app = express();

const PORT = process.env.PORT || 3001;

const DATA\_DIR = path.join(\_\_dirname, 'data');

// Helper: гарантируем наличие папки и файлов

function ensureData() {

  if (!fs.existsSync(DATA\_DIR)) fs.mkdirSync(DATA\_DIR, { recursive: true });

  const files = {

    users: 'users.json',

    orders: 'orders.json',

    cart: 'cart.json',

    favs: 'favs.json',

    compare: 'compare.json'

  };

  for (const k in files) {

    const p = path.join(DATA\_DIR, files[k]);

    if (!fs.existsSync(p)) fs.writeFileSync(p, '[]', 'utf8');

  }

}

ensureData();

function readJSON(filename) {

  const p = path.join(DATA\_DIR, filename);

  try {

    const raw = fs.readFileSync(p, 'utf8');

    return JSON.parse(raw || '[]');

  } catch (err) {

    console.error('readJSON error', filename, err);

    return [];

  }

}

function writeJSON(filename, data) {

  const p = path.join(DATA\_DIR, filename);

  fs.writeFileSync(p, JSON.stringify(data, null, 2), 'utf8');

}

// Middleware

app.use(cors());

app.use(express.json());

// --- Health

app.get('/', (req, res) => res.send({ ok: true }));

// ----------------- Users / Auth -----------------

// Register

app.post('/api/register', (req, res) => {

  try {

    const { email, password, firstName = '', lastName = '' } = req.body;

    if (!email || !password) return res.status(400).json({ error: 'Email and password required' });

    const users = readJSON('users.json');

    if (users.find(u => u.email === email)) return res.status(409).json({ error: 'User exists' });

    const newUser = {

      id: uuidv4(),

      email,

      password, // Для простоты: plain text. В реальном приложении — хэшировать.

      firstName,

      lastName,

      phone: '',

      address: '',

      role: 'Покупатель',

      createdAt: Date.now()

    };

    users.push(newUser);

    writeJSON('users.json', users);

    const safeUser = { ...newUser }; delete safeUser.password;

    res.json({ user: safeUser });

  } catch (err) {

    console.error(err); res.status(500).json({ error: 'Server error' });

  }

});

// Login

app.post('/api/login', (req, res) => {

  try {

    const { email, password } = req.body;

    const users = readJSON('users.json');

    const user = users.find(u => u.email === email && u.password === password);

    if (!user) return res.status(401).json({ error: 'Invalid credentials' });

    const safeUser = { ...user }; delete safeUser.password;

    res.json({ user: safeUser });

  } catch (err) {

    console.error(err); res.status(500).json({ error: 'Server error' });

  }

});

// Get current user by id

app.get('/api/users/:id', (req, res) => {

  try {

    const users = readJSON('users.json');

    const user = users.find(u => u.id === req.params.id);

    if (!user) return res.status(404).json({ error: 'User not found' });

    const safe = { ...user }; delete safe.password;

    res.json({ user: safe });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

// Update user

app.put('/api/users/:id', (req, res) => {

  try {

    const id = req.params.id;

    const payload = req.body;

    const users = readJSON('users.json');

    const idx = users.findIndex(u => u.id === id);

    if (idx === -1) return res.status(404).json({ error: 'User not found' });

    // Обновляем поля (не меняем пароль, если явно не передан)

    users[idx] = { ...users[idx], ...payload, id };

    writeJSON('users.json', users);

    const safe = { ...users[idx] }; delete safe.password;

    res.json({ user: safe });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

// List users (admin)

app.get('/api/users', (req, res) => {

  try {

    const users = readJSON('users.json').map(u => { const s = { ...u }; delete s.password; return s; });

    res.json({ users });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

// ----------------- Orders -----------------

// Create order

app.post('/api/orders', (req, res) => {

  try {

    const order = req.body; // ожидаем: { userId, items: [{name, price, qty}], total }

    if (!order || !order.userId) return res.status(400).json({ error: 'Invalid order' });

    const orders = readJSON('orders.json');

    const newOrder = { id: uuidv4(), ...order, createdAt: Date.now() };

    orders.push(newOrder);

    writeJSON('orders.json', orders);

    res.json({ order: newOrder });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

// Get orders by userId

app.get('/api/orders/:userId', (req, res) => {

  try {

    const all = readJSON('orders.json');

    const userOrders = all.filter(o => o.userId === req.params.userId);

    res.json({ orders: userOrders });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

// ----------------- Cart (server-side) -----------------

// Get cart for user

app.get('/api/cart/:userId', (req, res) => {

  try {

    const carts = readJSON('cart.json');

    const cart = carts.find(c => c.userId === req.params.userId) || { userId: req.params.userId, items: [] };

    res.json({ cart });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

// Update cart (replace)

app.put('/api/cart/:userId', (req, res) => {

  try {

    const userId = req.params.userId;

    const items = req.body.items || [];

    const carts = readJSON('cart.json');

    const idx = carts.findIndex(c => c.userId === userId);

    if (idx !== -1) {

      carts[idx].items = items;

    } else {

      carts.push({ userId, items });

    }

    writeJSON('cart.json', carts);

    res.json({ cart: carts.find(c => c.userId === userId) });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

// ----------------- Favorites -----------------

app.get('/api/favs/:userId', (req, res) => {

  try {

    const all = readJSON('favs.json');

    const entry = all.find(e => e.userId === req.params.userId) || { userId: req.params.userId, items: [] };

    res.json({ favs: entry.items });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

app.put('/api/favs/:userId', (req, res) => {

  try {

    const userId = req.params.userId;

    const items = req.body.items || [];

    const all = readJSON('favs.json');

    const idx = all.findIndex(e => e.userId === userId);

    if (idx !== -1) all[idx].items = items; else all.push({ userId, items });

    writeJSON('favs.json', all);

    res.json({ favs: items });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

// ----------------- Compare -----------------

app.get('/api/compare/:userId', (req, res) => {

  try {

    const all = readJSON('compare.json');

    const entry = all.find(e => e.userId === req.params.userId) || { userId: req.params.userId, items: [] };

    res.json({ compare: entry.items });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

app.put('/api/compare/:userId', (req, res) => {

  try {

    const userId = req.params.userId;

    const items = req.body.items || [];

    const all = readJSON('compare.json');

    const idx = all.findIndex(e => e.userId === userId);

    if (idx !== -1) all[idx].items = items; else all.push({ userId, items });

    writeJSON('compare.json', all);

    res.json({ compare: items });

  } catch (err) { console.error(err); res.status(500).json({ error: 'Server error' }); }

});

// ----------------- Start

app.listen(PORT, () => {

  console.log(`Server running at http://localhost:${PORT}`);

});

Листинг 2 – server.cjs

import express from 'express';

import fs from 'fs';

import path from 'path';

import { fileURLToPath } from 'url';

import cors from 'cors';

import bcrypt from 'bcrypt';

import jwt from 'jsonwebtoken';

import dotenv from 'dotenv';

// Load environment variables

dotenv.config();

const \_\_filename = fileURLToPath(import.meta.url);

const \_\_dirname = path.dirname(\_\_filename);

const app = express();

const PORT = 3001;

const DATA\_PATH = path.resolve(\_\_dirname, 'data', 'users.json');

// JWT Secret - в продакшене должен быть в переменных окружения

const JWT\_SECRET = process.env.JWT\_SECRET || 'supersecretkeyfordevelopment';

app.use(cors());

app.use(express.json());

// Middleware для проверки JWT токена

const authenticateToken = (req, res, next) => {

  const authHeader = req.headers['authorization'];

  const token = authHeader && authHeader.split(' ')[1]; // Bearer TOKEN

  if (!token) {

    return res.status(401).json({ error: 'Access token required' });

  }

  jwt.verify(token, JWT\_SECRET, (err, user) => {

    if (err) {

      return res.status(403).json({ error: 'Invalid or expired token' });

    }

    req.user = user;

    next();

  });

};

// Функция для безопасного разбора JSON

function safeJsonParse(data, defaultValue = null) {

  if (!data || data.length === 0) {

    return defaultValue;

  }

  try {

    return JSON.parse(data);

  } catch (error) {

    console.error('Ошибка парсинга JSON:', error);

    return defaultValue;

  }

}

app.get('/', (req, res) => {

  res.send('Сервер работает');

});

// Регистрация нового пользователя

app.post('/api/register', async (req, res) => {

  try {

    const { email, password, firstName, lastName } = req.body;

    // Проверяем, существует ли пользователь с таким email

    const raw = fs.existsSync(DATA\_PATH) ? fs.readFileSync(DATA\_PATH, 'utf-8') : '[]';

    const users = JSON.parse(raw);

    const existingUser = users.find(u => u.email === email);

    if (existingUser) {

      return res.status(409).json({ error: 'Пользователь с таким email уже существует' });

    }

    // Хешируем пароль

    const hashedPassword = await bcrypt.hash(password, 10);

    // Создаем нового пользователя

    const newUser = {

      id: Date.now().toString(), // генерируем уникальный ID

      email,

      password: hashedPassword, // сохраняем хешированный пароль

      firstName: firstName || '',

      lastName: lastName || '',

      role: 'user',

      createdAt: new Date().toISOString()

    };

    users.push(newUser);

    // Сохраняем в файл

    fs.writeFileSync(DATA\_PATH, JSON.stringify(users, null, 2));

    // Создаем JWT токен

    const token = jwt.sign(

      { id: newUser.id, email: newUser.email },

      JWT\_SECRET,

      { expiresIn: '7d' } // токен действителен 7 дней

    );

    // Возвращаем токен и данные пользователя (без пароля)

    res.json({

      success: true,

      token,

      user: {

        id: newUser.id,

        email: newUser.email,

        firstName: newUser.firstName,

        lastName: newUser.lastName,

        role: newUser.role,

        createdAt: newUser.createdAt

      }

    });

  } catch (err) {

    console.error('Ошибка регистрации:', err);

    res.status(500).json({ error: 'Ошибка сервера при регистрации' });

  }

});

// Вход пользователя

app.post('/api/login', async (req, res) => {

  try {

    const { email, password } = req.body;

    // Читаем пользователей из файла

    const raw = fs.existsSync(DATA\_PATH) ? fs.readFileSync(DATA\_PATH, 'utf-8') : '[]';

    const users = JSON.parse(raw);

    // Находим пользователя по email

    const user = users.find(u => u.email === email);

    if (!user) {

      return res.status(401).json({ error: 'Неверный email или пароль' });

    }

    // Проверяем пароль

    const isPasswordValid = await bcrypt.compare(password, user.password);

    if (!isPasswordValid) {

      return res.status(401).json({ error: 'Неверный email или пароль' });

    }

    // Создаем JWT токен

    const token = jwt.sign(

      { id: user.id, email: user.email },

      JWT\_SECRET,

      { expiresIn: '7d' }

    );

    // Возвращаем токен и данные пользователя (без пароля)

    res.json({

      success: true,

      token,

      user: {

        id: user.id,

        email: user.email,

        firstName: user.firstName,

        lastName: user.lastName,

        role: user.role,

        createdAt: user.createdAt

      }

    });

  } catch (err) {

    console.error('Ошибка входа:', err);

    res.status(500).json({ error: 'Ошибка сервера при входе' });

  }

});

// Получение профиля пользователя (защищенный маршрут)

app.get('/api/users/:id', authenticateToken, (req, res) => {

  try {

    const userId = req.params.id;

    // Проверяем, что пользователь запрашивает свой профиль

    if (req.user.id !== userId) {

      return res.status(403).json({ error: 'Доступ запрещен' });

    }

    const raw = fs.existsSync(DATA\_PATH) ? fs.readFileSync(DATA\_PATH, 'utf-8') : '[]';

    const users = JSON.parse(raw);

    const user = users.find(u => u.id === userId);

    if (!user) {

      return res.status(404).json({ error: 'Пользователь не найден' });

    }

    // Возвращаем данные пользователя без пароля

    res.json({

      user: {

        id: user.id,

        email: user.email,

        firstName: user.firstName,

        lastName: user.lastName,

        phone: user.phone,

        address: user.address,

        role: user.role,

        createdAt: user.createdAt

      }

    });

  } catch (err) {

    console.error('Ошибка получения профиля:', err);

    res.status(500).json({ error: 'Ошибка сервера' });

  }

});

// Обновление профиля пользователя (защищенный маршрут)

app.put('/api/users/:id', authenticateToken, (req, res) => {

  try {

    const userId = req.params.id;

    // Проверяем, что пользователь обновляет свой профиль

    if (req.user.id !== userId) {

      return res.status(403).json({ error: 'Доступ запрещен' });

    }

    const updateData = req.body;

    const raw = fs.existsSync(DATA\_PATH) ? fs.readFileSync(DATA\_PATH, 'utf-8') : '[]';

    const users = JSON.parse(raw);

    const userIndex = users.findIndex(u => u.id === userId);

    if (userIndex === -1) {

      return res.status(404).json({ error: 'Пользователь не найден' });

    }

    // Обновляем только разрешенные поля

    users[userIndex] = {

      ...users[userIndex],

      firstName: updateData.firstName || users[userIndex].firstName,

      lastName: updateData.lastName || users[userIndex].lastName,

      phone: updateData.phone || users[userIndex].phone,

      address: updateData.address || users[userIndex].address

    };

    // Сохраняем в файл

    fs.writeFileSync(DATA\_PATH, JSON.stringify(users, null, 2));

    // Возвращаем обновленные данные пользователя

    res.json({

      success: true,

      user: {

        id: users[userIndex].id,

        email: users[userIndex].email,

        firstName: users[userIndex].firstName,

        lastName: users[userIndex].lastName,

        phone: users[userIndex].phone,

        address: users[userIndex].address,

        role: users[userIndex].role,

        createdAt: users[userIndex].createdAt

      }

    });

  } catch (err) {

    console.error('Ошибка обновления профиля:', err);

    res.status(500).json({ error: 'Ошибка сервера' });

  }

});

// Обновление профиля пользователя (защищенный маршрут)

app.post('/api/save-user', authenticateToken, (req, res) => {

  try {

    const userId = req.user.id; // используем ID из токена

    const userData = req.body;

    const raw = fs.existsSync(DATA\_PATH) ? fs.readFileSync(DATA\_PATH, 'utf-8') : '[]';

    const users = JSON.parse(raw);

    const idx = users.findIndex(u => u.id === userId);

    if (idx !== -1) {

      // Обновляем только разрешенные поля

      users[idx] = {

        ...users[idx],

        firstName: userData.firstName || users[idx].firstName,

        lastName: userData.lastName || users[idx].lastName,

        phone: userData.phone || users[idx].phone,

        address: userData.address || users[idx].address

      };

    } else {

      // Если пользователя нет, добавляем его (в реальной ситуации такого быть не должно)

      users.push({

        id: userId,

        ...userData

      });

    }

    fs.writeFileSync(DATA\_PATH, JSON.stringify(users, null, 2));

    res.json({ success: true });

  } catch (err) {

    console.error('Ошибка сервера:', err);

    res.status(500).json({ error: 'Ошибка сервера' });

  }

});

// Маршруты для заказов

const ORDERS\_PATH = path.resolve(\_\_dirname, 'data', 'orders.json');

// Создание заказа

app.post('/api/orders', authenticateToken, (req, res) => {

  try {

    const userId = req.user.id;

    console.log('Создание заказа для пользователя:', userId);

    console.log('Тело запроса:', req.body);

    const orderData = { ...req.body, userId, id: Date.now().toString(), createdAt: new Date().toISOString(), status: 'pending' };

    console.log('Данные заказа:', orderData);

    const raw = fs.existsSync(ORDERS\_PATH) ? fs.readFileSync(ORDERS\_PATH, 'utf-8') : '[]';

    const orders = JSON.parse(raw);

    orders.push(orderData);

    fs.writeFileSync(ORDERS\_PATH, JSON.stringify(orders, null, 2));

    console.log('Заказ успешно записан в файл');

    res.json({ success: true, order: orderData });

  } catch (err) {

    console.error('Ошибка создания заказа:', err);

    res.status(500).json({ error: 'Ошибка сервера' });

  }

});

// Получение заказов пользователя

app.get('/api/orders/:userId', authenticateToken, (req, res) => {

  try {

    const userId = req.params.userId;

    // Проверяем, что пользователь запрашивает свои заказы

    if (req.user.id !== userId) {

      console.log('Попытка доступа к чужим заказам. Токен пользователя:', req.user.id, 'Запрашиваемый ID:', userId);

      return res.status(403).json({ error: 'Доступ запрещен' });

    }

    console.log('Получение заказов для пользователя:', userId);

    // Используем тот же путь, что и для записи заказов

    const raw = fs.existsSync(ORDERS\_PATH) ? fs.readFileSync(ORDERS\_PATH, 'utf-8') : '[]';

    const orders = JSON.parse(raw);

    // Фильтруем заказы по ID пользователя

    const userOrders = orders.filter(order => order.userId === userId);

    console.log('Найдено заказов для пользователя', userId, ':', userOrders.length);

    console.log('Заказы пользователя', userId, ':', userOrders);

    // Возвращаем заказы пользователя в формате { orders: [...] }, как ожидается в клиентском коде

    res.json({ orders: userOrders });

  } catch (error) {

    console.error('Ошибка получения заказов:', error);

    res.status(500).json({ error: 'Ошибка сервера при получении заказов' });

  }

});

// Маршруты для корзины

const CART\_PATH = path.resolve(\_\_dirname, 'data', 'cart.json');

app.get('/api/cart/:userId', authenticateToken, (req, res) => {

  try {

    const userId = req.params.userId;

    // Проверяем, что пользователь запрашивает свою корзину

    if (req.user.id !== userId) {

      return res.status(403).json({ error: 'Доступ запрещен' });

    }

    const raw = fs.existsSync(CART\_PATH) ? fs.readFileSync(CART\_PATH, 'utf-8') : '{}';

    const cart = JSON.parse(raw);

    res.json({ items: cart[userId] || [] });

  } catch (err) {

    console.error('Ошибка получения корзины:', err);

    res.status(500).json({ error: 'Ошибка сервера' });

  }

});

app.put('/api/cart/:userId', authenticateToken, (req, res) => {

  try {

    const userId = req.params.userId;

    // Проверяем, что пользователь обновляет свою корзину

    if (req.user.id !== userId) {

      return res.status(403).json({ error: 'Доступ запрещен' });

    }

    const { items } = req.body;

    const raw = fs.existsSync(CART\_PATH) ? fs.readFileSync(CART\_PATH, 'utf-8') : '{}';

    const cart = JSON.parse(raw);

    cart[userId] = items || [];

    fs.writeFileSync(CART\_PATH, JSON.stringify(cart, null, 2));

    res.json({ success: true });

  } catch (err) {

    console.error('Ошибка обновления корзины:', err);

    res.status(500).json({ error: 'Ошибка сервера' });

  }

});

// Маршруты для избранного

const FAVS\_PATH = path.resolve(\_\_dirname, 'data', 'favs.json');

app.get('/api/favs/:userId', authenticateToken, (req, res) => {

  try {

    const userId = req.params.userId;

    // Проверяем, что пользователь запрашивает свои избранные

    if (req.user.id !== userId) {

      return res.status(403).json({ error: 'Доступ запрещен' });

    }

    const raw = fs.existsSync(FAVS\_PATH) ? fs.readFileSync(FAVS\_PATH, 'utf-8') : '{}';

    const favs = JSON.parse(raw);

    res.json({ items: favs[userId] || [] });

  } catch (err) {

    console.error('Ошибка получения избранного:', err);

    res.status(500).json({ error: 'Ошибка сервера' });

  }

});

app.put('/api/favs/:userId', authenticateToken, (req, res) => {

  try {

    const userId = req.params.userId;

    // Проверяем, что пользователь обновляет свои избранные

    if (req.user.id !== userId) {

      return res.status(403).json({ error: 'Доступ запрещен' });

    }

    const { items } = req.body;

    const raw = fs.existsSync(FAVS\_PATH) ? fs.readFileSync(FAVS\_PATH, 'utf-8') : '{}';

    const favs = JSON.parse(raw);

    favs[userId] = items || [];

    fs.writeFileSync(FAVS\_PATH, JSON.stringify(favs, null, 2));

    res.json({ success: true });

  } catch (err) {

    console.error('Ошибка обновления избранного:', err);

    res.status(500).json({ error: 'Ошибка сервера' });

  }

});

// Маршруты для сравнения

const COMPARE\_PATH = path.resolve(\_\_dirname, 'data', 'compare.json');

app.get('/api/compare/:userId', authenticateToken, (req, res) => {

  try {

    const userId = req.params.userId;

    // Проверяем, что пользователь запрашивает свои товары для сравнения

    if (req.user.id !== userId) {

      return res.status(403).json({ error: 'Доступ запрещен' });

    }

    const raw = fs.existsSync(COMPARE\_PATH) ? fs.readFileSync(COMPARE\_PATH, 'utf-8') : '{}';

    const compare = JSON.parse(raw);

    res.json({ items: compare[userId] || [] });

  } catch (err) {

    console.error('Ошибка получения сравнения:', err);

    res.status(500).json({ error: 'Ошибка сервера' });

  }

});

app.put('/api/compare/:userId', authenticateToken, (req, res) => {

  try {

    const userId = req.params.userId;

    // Проверяем, что пользователь обновляет свои товары для сравнения

    if (req.user.id !== userId) {

      return res.status(403).json({ error: 'Доступ запрещен' });

    }

    const { items } = req.body;

    const raw = fs.existsSync(COMPARE\_PATH) ? fs.readFileSync(COMPARE\_PATH, 'utf-8') : '{}';

    const compare = JSON.parse(raw);

    compare[userId] = items || [];

    fs.writeFileSync(COMPARE\_PATH, JSON.stringify(compare, null, 2));

    res.json({ success: true });

  } catch (err) {

    console.error('Ошибка обновления сравнения:', err);

    res.status(500).json({ error: 'Ошибка сервера' });

  }

});

app.listen(PORT, () => {

  console.log(`Server running at http://localhost:${PORT}`);

});

Листинг 3 – server.js

import React from 'react';

import { Routes, Route } from 'react-router-dom';

import { NotificationProvider } from '../contexts/NotificationContext';

import Header from '../components/Header';

import Home from '../pages/Home';

import Catalog from '../pages/Catalog';

import Cart from '../pages/Cart';

import Profile from '../pages/Profile';

import Favorites from '../pages/Favourites';

import Compare from '../pages/Compare';

import OrderDetails from '../pages/OrderDetails';

import PrivateRoute from '../components/PrivateRoute';

import { useSyncWithServer } from '../utils/syncUtils';

function App() {

  useSyncWithServer(); // Initialize sync with server

  return (

    <NotificationProvider>

      <Header />

      <Routes>

        <Route path="/" element={<Home />} />

        <Route path="/catalog" element={<Catalog />} />

        <Route path="/cart" element={

          <PrivateRoute>

            <Cart />

          </PrivateRoute>

        } />

        <Route path="/favorites" element={

          <PrivateRoute>

            <Favorites />

          </PrivateRoute>

        } />

        <Route path="/profile" element={

          <PrivateRoute>

            <Profile />

          </PrivateRoute>

        } />

        <Route path="/compare" element={

          <PrivateRoute>

            <Compare />

          </PrivateRoute>

        } />

        <Route path="/order/:orderId" element={

          <PrivateRoute>

            <OrderDetails />

          </PrivateRoute>

        } />

      </Routes>

    </NotificationProvider>

  );

}

export default App;

Листинг 4 – App.jsx

import { configureStore } from '@reduxjs/toolkit';

import { persistStore, persistReducer } from 'redux-persist';

import storage from 'redux-persist/lib/storage';

import authReducer from '../store/authSlice';

import productsReducer from '../features/products/productsSlice';

import cartReducer from '../features/cart/cartSlice';

import ordersReducer from '../store/orderSlice';

import { authMiddleware, checkAuthTokenOnLoad } from '../utils/authMiddleware';

const authPersistConfig = {

  key: 'auth',

  storage,

};

const productsPersistConfig = {

  key: 'products',

  storage,

};

const cartPersistConfig = {

  key: 'cart',

  storage,

};

const store = configureStore({

  reducer: {

    auth: persistReducer(authPersistConfig, authReducer),

    products: persistReducer(productsPersistConfig, productsReducer),

    cart: persistReducer(cartPersistConfig, cartReducer),

    orders: ordersReducer,

  },

  middleware: (getDefaultMiddleware) =>

    getDefaultMiddleware({

      serializableCheck: false,

    }).concat(authMiddleware),

});

// Проверяем токен при загрузке приложения и обновляем состояние, если нужно

const user = checkAuthTokenOnLoad();

if (user && store.getState().auth.user === null) {

  // Если у нас есть пользователь в localStorage, но Redux не знает о нем,

  // возможно, приложение было перезагружено и нужно обновить состояние

  store.dispatch({ type: 'auth/setUser', payload: user });

}

export const persistor = persistStore(store);

export default store;

Листинг 5 – store.jsx

import React, { useState, useMemo } from 'react';

import { useSelector, useDispatch } from 'react-redux';

import {

  setCategoryFilter,

  setBrandFilter,

  setPriceFilter,

  setSearchFilter,

  resetFilters

} from '../features/products/productsSlice';

import styles from '../styles/Filters.module.css';

function Filters() {

  const dispatch = useDispatch();

  const filters = useSelector(state => state.products.filters);

  const products = useSelector(state => state.products.items);

  // Получаем уникальные категории и бренды из товаров

  const categories = useMemo(() => [...new Set(products.map(p => p.category))], [products]);

  const brands = useMemo(() => [...new Set(products.map(p => p.brand))], [products]);

  // Стейты для локального управления формой

  const [tempMinPrice, setTempMinPrice] = useState(filters.minPrice);

  const [tempMaxPrice, setTempMaxPrice] = useState(filters.maxPrice);

  const [tempSearch, setTempSearch] = useState(filters.search);

  // Обработчики изменений

  const handleCategoryChange = (e) => {

    dispatch(setCategoryFilter(e.target.value));

  };

  const handleBrandChange = (e) => {

    dispatch(setBrandFilter(e.target.value));

  };

  const handlePriceChange = () => {

    dispatch(setPriceFilter({ min: tempMinPrice, max: tempMaxPrice }));

  };

  const handleSearchChange = () => {

    dispatch(setSearchFilter(tempSearch));

  };

  const handleResetFilters = () => {

    dispatch(resetFilters());

    setTempMinPrice(0);

    setTempMaxPrice(10000);

    setTempSearch('');

  };

  const handleKeyPress = (e) => {

    if (e.key === 'Enter') {

      handleSearchChange();

    }

  };

  return (

    <div className={styles.filters}>

      <h3>Фильтры</h3>

      {/\* Поиск \*/}

      <div className={styles.filterGroup}>

        <label>Поиск:</label>

        <div className={styles.searchContainer}>

          <input

            type="text"

            value={tempSearch}

            onChange={(e) => setTempSearch(e.target.value)}

            onKeyPress={handleKeyPress}

            placeholder="Поиск по названию, аниме или персонажу"

          />

          <button onClick={handleSearchChange}>Найти</button>

        </div>

      </div>

      {/\* Категории \*/}

      <div className={styles.filterGroup}>

        <label>Категория:</label>

        <select value={filters.category} onChange={handleCategoryChange}>

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

          {categories.map(category => (

            <option key={category} value={category}>

              {getCategoryName(category)}

            </option>

          ))}

        </select>

      </div>

      {/\* Бренды \*/}

      <div className={styles.filterGroup}>

        <label>Бренд:</label>

        <select value={filters.brand} onChange={handleBrandChange}>

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

          {brands.map(brand => (

            <option key={brand} value={brand}>{brand}</option>

          ))}

        </select>

      </div>

      {/\* Цена \*/}

      <div className={styles.filterGroup}>

        <label>Цена:</label>

        <div className={styles.priceInputs}>

          <input

            type="number"

            value={tempMinPrice}

            onChange={(e) => setTempMinPrice(Number(e.target.value))}

            placeholder="Мин"

          />

          <span>-</span>

          <input

            type="number"

            value={tempMaxPrice}

            onChange={(e) => setTempMaxPrice(Number(e.target.value))}

            placeholder="Макс"

          />

          <button onClick={handlePriceChange}>Применить</button>

        </div>

      </div>

      {/\* Кнопка сброса \*/}

      <div className={styles.filterGroup}>

        <button onClick={handleResetFilters} className={styles.resetButton}>

          Сбросить фильтры

        </button>

      </div>

    </div>

  );

}

// Вспомогательная функция для получения понятного названия категории

const getCategoryName = (category) => {

  const categoryNames = {

    'figurines': 'Фигурки',

    'posters': 'Плакаты',

    'books': 'Книги',

    'jewelry': 'Украшения',

    'clothing': 'Одежда',

    'accessories': 'Аксессуары',

    'cosplay': 'Косплей',

    'sweets': 'Сладости'

  };

  return categoryNames[category] || category;

};

export default Filters;

Листинг 6 – filters.jsx

import React, { useState, useEffect } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { useSelector, useDispatch } from 'react-redux';

import { loadCartFromServer } from '../features/cart/cartSlice';

import { loadFavsFromServer, loadCompareFromServer } from '../features/products/productsSlice';

import { selectUser, fetchUserProfile } from '../store/authSlice';

import { fetchProducts } from '../features/products/productsSlice';

import styles from '../styles/Header.module.css';

import logo from '../assets/logo.png';

import cartIcon from '../assets/icons/cart.png';

import heartIcon from '../assets/icons/heart.png';

import compareIcon from '../assets/icons/compare.png';

import userIcon from '../assets/icons/user.png';

import Tooltip from './Tooltip';

import LoginRegisterModal from './LoginRegisterModal';

import { useNotification } from '../contexts/NotificationContext';

function Header() {

  const dispatch = useDispatch();

  const authUser = useSelector(selectUser);

  const allProducts = useSelector(state => state.products.items);

  const cartItems = useSelector(state => state.cart.items);

  const favoriteItems = useSelector(state => state.products.favorites);

  const compareItems = useSelector(state => state.products.compare);

  const [authModalOpen, setAuthModalOpen] = useState(false);

  const [searchQuery, setSearchQuery] = useState('');

  const [searchResults, setSearchResults] = useState([]);

  const [showSearchResults, setShowSearchResults] = useState(false);

  const navigate = useNavigate();

  const { addNotification } = useNotification();

  useEffect(() => {

    // Load user data from server if authenticated

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

    if (authUser?.id && token) {

      // Load cart, favorites, and comparison from server

      dispatch(loadCartFromServer(authUser.id));

      dispatch(loadFavsFromServer(authUser.id));

      dispatch(loadCompareFromServer(authUser.id));

      dispatch(fetchUserProfile(authUser.id));

    }

  }, [authUser, dispatch]);

  useEffect(() => {

    // Load all products for search

    dispatch(fetchProducts());

  }, [dispatch]);

  useEffect(() => {

    if (searchQuery.trim() === '') {

      setSearchResults([]);

      return;

    }

    // Filter products based on search query

    const filteredProducts = allProducts.filter(item =>

      item.title.toLowerCase().includes(searchQuery.toLowerCase()) ||

      (item.animeSeries && item.animeSeries.toLowerCase().includes(searchQuery.toLowerCase())) ||

      (item.character && item.character.toLowerCase().includes(searchQuery.toLowerCase()))

    ).slice(0, 5); // Limit to 5 results

    // Get unique categories

    const categoriesSet = new Set();

    allProducts.forEach(item => {

      if ((item.category && item.category.toLowerCase().includes(searchQuery.toLowerCase())) ||

          (item.animeSeries && item.animeSeries.toLowerCase().includes(searchQuery.toLowerCase()))) {

        const category = item.category || item.animeSeries;

        if (category) {

          categoriesSet.add(category);

        }

      }

    });

    const categories = Array.from(categoriesSet).slice(0, 3); // Limit to 3 categories

    // Combine products and categories in results

    const results = [

      ...filteredProducts.map(item => ({ type: 'product', ...item })),

      ...categories.map(category => ({ type: 'category', name: category }))

    ];

    setSearchResults(results);

  }, [searchQuery, allProducts]);

  const handleSearchChange = (e) => {

    setSearchQuery(e.target.value);

    setShowSearchResults(true);

  };

  const handleSearchSubmit = (e) => {

    e.preventDefault();

    if (searchQuery.trim()) {

      navigate(`/catalog?search=${encodeURIComponent(searchQuery)}`);

      setSearchQuery('');

      setShowSearchResults(false);

    }

  };

  const handleResultClick = (result) => {

    if (result.type === 'product') {

      // For now, just navigate to catalog with search term

      navigate(`/catalog?search=${encodeURIComponent(result.title)}`);

    } else if (result.type === 'category') {

      navigate(`/catalog?category=${encodeURIComponent(result.name)}`);

    }

    setSearchQuery('');

    setShowSearchResults(false);

  };

  const isAuthenticated = Boolean(authUser?.id);

  const openOrNavigate = (path) => {

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

    if (!isAuthenticated || !token) {

      setAuthModalOpen(true);

    } else {

      navigate(path);

    }

  };

  const handleCartClick = () => {

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

    if (!isAuthenticated || !token) {

      setAuthModalOpen(true);

    } else {

      navigate('/cart');

      if (cartItems.length > 0) {

        addNotification(`Корзина: ${cartItems.length} ${cartItems.length === 1 ? 'товар' : cartItems.length < 5 ? 'товара' : 'товаров'}`, 'info', 2000);

      } else {

        addNotification('Корзина пуста', 'info', 2000);

      }

    }

  };

  const handleFavoritesClick = () => {

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

    if (!isAuthenticated || !token) {

      setAuthModalOpen(true);

    } else {

      navigate('/favorites');

      if (favoriteItems.length > 0) {

        addNotification(`Избранное: ${favoriteItems.length} ${favoriteItems.length === 1 ? 'товар' : favoriteItems.length < 5 ? 'товара' : 'товаров'}`, 'info', 2000);

      } else {

        addNotification('Избранное пусто', 'info', 2000);

      }

    }

  };

  const handleCompareClick = () => {

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

    if (!isAuthenticated || !token) {

      setAuthModalOpen(true);

    } else {

      navigate('/compare');

      if (compareItems.length > 0) {

        addNotification(`Сравнение: ${compareItems.length} ${compareItems.length === 1 ? 'товар' : compareItems.length < 5 ? 'товара' : 'товаров'}`, 'info', 2000);

      } else {

        addNotification('Нет товаров для сравнения', 'info', 2000);

      }

    }

  };

  return (

    <>

      <header className={styles.header}>

        <div className={styles.logoBlock}>

          <Link to="/" className={styles.logoBlock}>

            <img src={logo} alt="Anime Shop Logo" className={styles.logo} />

          </Link>

        </div>

        <div className={styles.searchBlock}>

          <form onSubmit={handleSearchSubmit} className={styles.searchForm}>

            <input

              type="text"

              placeholder="Поиск товаров..."

              className={styles.searchInput}

              value={searchQuery}

              onChange={handleSearchChange}

              onFocus={() => setShowSearchResults(true)}

            />

            <button type="submit" className={styles.searchButton}>🔍</button>

          </form>

          {showSearchResults && searchResults.length > 0 && (

            <div className={styles.searchResults}>

              {searchResults.map((result, index) => (

                <div

                  key={index}

                  className={styles.searchResultItem}

                  onClick={() => handleResultClick(result)}

                >

                  {result.type === 'product' ? (

                    <div className={styles.productResult}>

                      <img src={result.image} alt={result.title} className={styles.resultImage} />

                      <div className={styles.resultInfo}>

                        <div className={styles.resultTitle}>{result.title}</div>

                        <div className={styles.resultPrice}>{result.price} BYN</div>

                      </div>

                    </div>

                  ) : (

                    <div className={styles.categoryResult}>

                      📁 {result.name}

                    </div>

                  )}

                </div>

              ))}

            </div>

          )}

        </div>

        <div className={styles.actions}>

          <Tooltip text="Корзина">

            <button className={styles.iconButton} onClick={handleCartClick}>

              <img src={cartIcon} alt="Корзина" />

              {cartItems.length > 0 && (

                <span className={styles.badge}>{cartItems.reduce((total, item) => total + item.quantity, 0)}</span>

              )}

            </button>

          </Tooltip>

          <Tooltip text="Избранное">

            <button className={styles.iconButton} onClick={handleFavoritesClick}>

              <img

                src={heartIcon}

                alt="Избранное"

                className={favoriteItems.length > 0 ? styles.heartFilled : ''}

              />

              {favoriteItems.length > 0 && (

                <span className={styles.badge}>{favoriteItems.length}</span>

              )}

            </button>

          </Tooltip>

          <Tooltip text="Сравнение">

            <button className={styles.iconButton} onClick={handleCompareClick}>

              <img src={compareIcon} alt="Сравнение" />

              {compareItems.length > 0 && (

                <span className={styles.badge}>{compareItems.length}</span>

              )}

            </button>

          </Tooltip>

          <Tooltip text={isAuthenticated ? "Вы вошли" : "Профиль"}>

            <button className={styles.iconButton} onClick={() => openOrNavigate('/profile')}>

              <img src={userIcon} alt="Профиль" />

              {isAuthenticated && <span className={styles.authMark}></span>}

            </button>

          </Tooltip>

        </div>

      </header>

      <LoginRegisterModal

        visible={authModalOpen}

        onClose={() => setAuthModalOpen(false)}

        onLogin={(user) => {

          // After login, the Redux state will be updated automatically

          setAuthModalOpen(false);

        }}

        />

    </>

  );

}

export default Header;

Листинг 7 – header.jsx

import React, { useState } from 'react';

import { useDispatch } from 'react-redux';

import { registerUser, login } from '../store/authSlice';

import styles from '../styles/LoginRegisterModal.module.css';

function LoginRegisterModal({ visible, onClose, onLogin }){

  const dispatch = useDispatch();

  const [mode, setMode] = useState('login');

  const [form, setForm] = useState({

    email: '',

    password: '',

    firstName: '',

    lastName: ''

  });

  const [message, setMessage] = useState(null);

  const [loading, setLoading] = useState(false);

  if (!visible) return null;

  const handleChange = (e) => {

    setForm(prev => ({ ...prev, [e.target.name]: e.target.value }));

  };

  const handleLogin = async (e) => {

    e.preventDefault();

    setMessage(null);

    setLoading(true);

    try {

      const result = await dispatch(login(form));

      if (result.payload) {

        if (onLogin) onLogin(result.payload);

        setMessage('Вы успешно вошли');

        setTimeout(() => {

          setLoading(false);

          onClose();

        }, 700);

      } else {

        setMessage(result.error?.message || 'Ошибка входа');

        setLoading(false);

      }

    } catch (error) {

      console.error('Ошибка входа:', error);

      setMessage('Ошибка входа');

      setLoading(false);

    }

  };

  const handleRegister = async (e) => {

    e.preventDefault();

    setMessage(null);

    setLoading(true);

    try {

      const result = await dispatch(registerUser(form));

      if (result.payload) {

        setMessage('Регистрация успешна. Теперь войдите.');

        setMode('login');

        setLoading(false);

      } else {

        setMessage(result.error?.message || 'Ошибка регистрации');

        setLoading(false);

      }

    } catch (error) {

      console.error('Ошибка регистрации:', error);

      setMessage('Ошибка регистрации');

      setLoading(false);

    }

  };

  return (

    <div className={styles.overlay} onClick={onClose}>

      <div className={styles.modal} onClick={(e) => e.stopPropagation()}>

        <button className={styles.close} onClick={onClose}>✕</button>

        <div className={styles.switch}>

          <button

            className={mode === 'login' ? styles.active : ''}

            onClick={() => { setMode('login'); setMessage(null); }}

          >

            Вход

          </button>

          <button

            className={mode === 'register' ? styles.active : ''}

            onClick={() => { setMode('register'); setMessage(null); }}

          >

            Регистрация

          </button>

        </div>

        {message && <div className={styles.message}>{message}</div>}

        {mode === 'login' ? (

          <form onSubmit={handleLogin} className={styles.form}>

            <label>Email<input name="email" value={form.email} onChange={handleChange} required /></label>

            <label>Пароль<input type="password" name="password" value={form.password} onChange={handleChange} required /></label>

            <button type="submit" className={styles.submit} disabled={loading}>

              {loading ? 'Вхожу...' : 'Войти'}

            </button>

          </form>

        ) : (

          <form onSubmit={handleRegister} className={styles.form}>

            <label>Email<input name="email" value={form.email} onChange={handleChange} required /></label>

            <label>Пароль<input type="password" name="password" value={form.password} onChange={handleChange} required /></label>

            <label>Имя<input name="firstName" value={form.firstName} onChange={handleChange} /></label>

            <label>Фамилия<input name="lastName" value={form.lastName} onChange={handleChange} /></label>

            <button type="submit" className={styles.submit} disabled={loading}>

              {loading ? 'Регистрация...' : 'Зарегистрироваться'}

            </button>

          </form>

        )}

      </div>

    </div>

  );

}

export default LoginRegisterModal;

Листинг 8 – LoginRegisterModal.jsx

import React, { useEffect } from 'react';

import styles from '../styles/Notification.module.css';

const Notification = ({ message, type = 'info', onClose, duration = 3000 }) => {

  useEffect(() => {

    const timer = setTimeout(() => {

      onClose();

    }, duration);

    return () => {

      clearTimeout(timer);

    };

  }, [onClose, duration]);

  const getNotificationClass = () => {

    switch (type) {

      case 'success':

        return styles.success;

      case 'error':

        return styles.error;

      case 'warning':

        return styles.warning;

      default:

        return styles.info;

    }

  };

  return (

    <div className={`${styles.notification} ${getNotificationClass()}`}>

      <span className={styles.message}>{message}</span>

      <button className={styles.closeButton} onClick={onClose}>×</button>

    </div>

  );

};

export default Notification;

Листинг 9 – Notification.jsx

import React from 'react';

import { Navigate } from 'react-router-dom';

import { useSelector } from 'react-redux';

import { selectUser } from '../store/authSlice';

const PrivateRoute = ({ children }) => {

  const authUser = useSelector(selectUser);

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

  // Если нет токена или пользователя, перенаправляем на главную

  if (!token || !authUser) {

    return <Navigate to="/" replace />;

  }

  return children;

};

export default PrivateRoute;

Листинг 10 – PrivateRoute.jsx

import React, { useState } from 'react';

import { useDispatch, useSelector } from 'react-redux';

import styles from '../styles/ProductCard.module.css';

import { addToCart } from '../features/cart/cartSlice';

import { addToFavorite, removeFromFavorite, addToCompare, removeFromCompare } from '../features/products/productsSlice';

import { selectUser } from '../store/authSlice';

import { useNotification } from '../contexts/NotificationContext';

function ProductCard({ product }) {

  const dispatch = useDispatch();

  const user = useSelector(selectUser);

  const [showDetails, setShowDetails] = useState(false);

  const [quantity, setQuantity] = useState(1);

  const { addNotification } = useNotification();

  const favorites = useSelector(state => state.products.favorites);

  const compareItems = useSelector(state => state.products.compare);

  const isFavorite = favorites.some(item => item.id === product.id);

  const isInCompare = compareItems.some(item => item.id === product.id);

  const handleAddToCart = () => {

    console.log('User clicked: Add to Cart', product.title);

    dispatch(addToCart({ ...product, quantity }));

    // Show notification

    addNotification(`✓ Товар "${product.title}" добавлен в корзину!`, 'success', 3000);

    // The useSyncWithServer hook will handle saving to server automatically

  };

  const handleToggleFavorite = () => {

    console.log('User clicked: Toggle Favorite', product.title);

    if (isFavorite) {

      dispatch(removeFromFavorite(product.id));

      addNotification(`❤️ Товар "${product.title}" удален из избранного`, 'info', 2000);

    } else {

      dispatch(addToFavorite(product));

      addNotification(`❤️ Товар "${product.title}" добавлен в избранное!`, 'success', 2000);

    }

    // The useSyncWithServer hook will handle saving to server automatically

  };

  const handleToggleCompare = () => {

    console.log('User clicked: Toggle Compare', product.title);

    if (isInCompare) {

      dispatch(removeFromCompare(product.id));

      addNotification(`⚖️ Товар "${product.title}" удален из сравнения`, 'info', 2000);

    } else {

      if (compareItems.length >= 4) {

        addNotification('⚠️ Нельзя добавить более 4 товаров для сравнения', 'warning', 3000);

        return;

      }

      dispatch(addToCompare(product));

      addNotification(`⚖️ Товар "${product.title}" добавлен для сравнения!`, 'success', 2000);

    }

    // The useSyncWithServer hook will handle saving to server automatically

  };

  const toggleDetails = () => {

    console.log('User clicked: Toggle Details');

    setShowDetails(!showDetails);

  };

  return (

    <div className={styles.card}>

      <img src={product.image} alt={product.title} className={styles.image} />

      <div className={styles.info}>

        <h3 className={styles.title}>{product.title}</h3>

        <p className={styles.price}>{product.price} BYN</p>

        <p className={styles.category}>{getCategoryName(product.category)}</p>

        <p className={styles.animeSeries}>Аниме: {product.animeSeries}</p>

        {showDetails && (

          <div className={styles.details}>

            <p><strong>Персонаж:</strong> {product.character}</p>

            <p><strong>Бренд:</strong> {product.brand}</p>

            <p><strong>Материал:</strong> {product.material}</p>

            <p><strong>Размер:</strong> {product.size}</p>

            <p><strong>Возраст:</strong> {product.ageRestriction}</p>

            <p><strong>Рейтинг:</strong> {product.rating}</p>

            <p><strong>Описание:</strong> {product.description}</p>

          </div>

        )}

        <div className={styles.buttonContainer}>

          <div className={styles.quantityControl}>

            <button

              onClick={() => setQuantity(Math.max(1, quantity - 1))}

              className={styles.quantityBtn}

            >

              -

            </button>

            <span className={styles.quantity}>{quantity}</span>

            <button

              onClick={() => setQuantity(quantity + 1)}

              className={styles.quantityBtn}

            >

              +

            </button>

          </div>

          <button onClick={handleAddToCart} className={styles.button}>

            Добавить в корзину

          </button>

          <div className={styles.actionButtons}>

            <button

              onClick={handleToggleFavorite}

              className={`${styles.actionButton} ${isFavorite ? styles.filled : ''}`}

            >

              {isFavorite ? '❤️' : '🤍'}

            </button>

            <button

              onClick={handleToggleCompare}

              className={`${styles.actionButton} ${isInCompare ? styles.filled : ''}`}

            >

              {isInCompare ? '⚖️' : '⚖️'}

            </button>

          </div>

          <button onClick={toggleDetails} className={styles.detailsButton}>

            {showDetails ? 'Скрыть' : 'Подробнее'}

          </button>

        </div>

      </div>

    </div>

  );

}

// Вспомогательная функция для получения понятного названия категории

const getCategoryName = (category) => {

  const categoryNames = {

    'figurines': 'Фигурки',

    'posters': 'Плакаты',

    'books': 'Книги',

    'jewelry': 'Украшения',

    'clothing': 'Одежда',

    'accessories': 'Аксессуары',

    'cosplay': 'Косплей',

    'sweets': 'Сладости'

  };

  return categoryNames[category] || category;

};

export default ProductCard;

Листинг 11 – ProductCard.jsx

import React from 'react';

import styles from '../styles/Tooltip.module.css';

function Tooltip({ text, children }) {

  return (

    <div className={styles.wrapper}>

      {children}

      <span className={styles.tooltip}>{text}</span>

    </div>

  );

}

export default Tooltip;

Листинг 12 – tooltip.jsx

import React, { createContext, useContext, useState } from 'react';

import Notification from '../components/Notification';

const NotificationContext = createContext();

export const useNotification = () => {

  const context = useContext(NotificationContext);

  if (!context) {

    throw new Error('useNotification must be used within a NotificationProvider');

  }

  return context;

};

export const NotificationProvider = ({ children }) => {

  const [notifications, setNotifications] = useState([]);

  const addNotification = (message, type = 'info', duration = 3000) => {

    const id = Date.now() + Math.random();

    setNotifications(prev => [

      ...prev,

      { id, message, type, duration }

    ]);

  };

  const removeNotification = (id) => {

    setNotifications(prev => prev.filter(notification => notification.id !== id));

  };

  return (

    <NotificationContext.Provider value={{ addNotification }}>

      {children}

      <div style={{ position: 'fixed', top: 0, right: 0, zIndex: 10000 }}>

        {notifications.map(({ id, message, type, duration }) => (

          <Notification

            key={id}

            message={message}

            type={type}

            duration={duration}

            onClose={() => removeNotification(id)}

          />

        ))}

      </div>

    </NotificationContext.Provider>

  );

};

Листинг 15 – NotificationContext.jsx

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

import { getCart, updateCart } from '../../utils/api';

// Async thunks for server sync

export const loadCartFromServer = createAsyncThunk(

  'cart/loadFromServer',

  async (userId, { rejectWithValue }) => {

    try {

      console.log('Loading cart from server for user:', userId);

      const response = await getCart(userId);

      console.log('Loaded cart items:', response.items);

      return response.items;

    } catch (error) {

      console.error('Error loading cart from server:', error);

      // The token should have been removed by api.js, component will handle the auth state

      if (error.status === 401 || error.status === 403) {

        // Token was removed by api.js, let the component handle the state change

      }

      return rejectWithValue(error.body?.error || 'Ошибка загрузки корзины');

    }

  }

);

export const saveCartToServer = createAsyncThunk(

  'cart/saveToServer',

  async ({ userId, items }, { rejectWithValue }) => {

    try {

      console.log('Saving cart to server for user:', userId, 'items:', items);

      await updateCart(userId, items);

      console.log('Successfully saved cart to server');

      return items;

    } catch (error) {

      console.error('Error saving cart to server:', error);

      // The token should have been removed by api.js, component will handle the auth state

      if (error.status === 401 || error.status === 403) {

        // Token was removed by api.js, let the component handle the state change

      }

      return rejectWithValue(error.body?.error || 'Ошибка сохранения корзины');

    }

  }

);

const cartSlice = createSlice({

  name: 'cart',

  initialState: { items: [] },

  reducers: {

    addToCart(state, action) {

      console.log('User action: addToCart', action.payload);

      const existingItem = state.items.find(item => item.id === action.payload.id);

      if (existingItem) {

        existingItem.quantity += action.payload.quantity || 1;

      } else {

        state.items.push({ ...action.payload, quantity: action.payload.quantity || 1 });

      }

    },

    removeFromCart(state, action) {

      console.log('User action: removeFromCart', action.payload);

      state.items = state.items.filter(item => item.id !== action.payload);

    },

    updateQuantity(state, action) {

      console.log('User action: updateQuantity', action.payload);

      const { id, quantity } = action.payload;

      const existingItem = state.items.find(item => item.id === id);

      if (existingItem) {

        existingItem.quantity = quantity;

        if (quantity <= 0) {

          state.items = state.items.filter(item => item.id !== id);

        }

      }

    },

    clearCart(state) {

      console.log('User action: clearCart');

      state.items = [];

    },

    setCartItems(state, action) {

      state.items = action.payload;

    }

  },

  extraReducers: (builder) => {

    builder

      .addCase(loadCartFromServer.fulfilled, (state, action) => {

        state.items = action.payload;

      })

      .addCase(saveCartToServer.fulfilled, (state, action) => {

        state.items = action.payload;

      });

  }

});

export const { addToCart, removeFromCart, updateQuantity, clearCart, setCartItems } = cartSlice.actions;

export default cartSlice.reducer;

Листинг 17 – CartSlice.js

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

import { getFavs, updateFavs, getCompare, updateCompare } from '../../utils/api';

export const fetchProducts = createAsyncThunk('products/fetch', async () => {

  const res = await fetch('/api/anime\_products.json');

  return await res.json();

});

// Async thunks for server sync

export const loadFavsFromServer = createAsyncThunk(

  'products/loadFavsFromServer',

  async (userId, { rejectWithValue }) => {

    try {

      console.log('Loading favorites from server for user:', userId);

      const response = await getFavs(userId);

      console.log('Loaded favorites:', response.items);

      return response.items;

    } catch (error) {

      console.error('Error loading favorites from server:', error);

      // The token should have been removed by api.js, component will handle the auth state

      if (error.status === 401 || error.status === 403) {

        // Token was removed by api.js, let the component handle the state change

      }

      return rejectWithValue(error.body?.error || 'Ошибка загрузки избранного');

    }

  }

);

export const saveFavsToServer = createAsyncThunk(

  'products/saveFavsToServer',

  async ({ userId, items }, { rejectWithValue }) => {

    try {

      console.log('Saving favorites to server for user:', userId, 'items:', items);

      await updateFavs(userId, items);

      console.log('Successfully saved favorites to server');

      return items;

    } catch (error) {

      console.error('Error saving favorites to server:', error);

      // The token should have been removed by api.js, component will handle the auth state

      if (error.status === 401 || error.status === 403) {

        // Token was removed by api.js, let the component handle the state change

      }

      return rejectWithValue(error.body?.error || 'Ошибка сохранения избранного');

    }

  }

);

export const loadCompareFromServer = createAsyncThunk(

  'products/loadCompareFromServer',

  async (userId, { rejectWithValue }) => {

    try {

      console.log('Loading compare from server for user:', userId);

      const response = await getCompare(userId);

      console.log('Loaded compare items:', response.items);

      return response.items;

    } catch (error) {

      console.error('Error loading compare from server:', error);

      // The token should have been removed by api.js, component will handle the auth state

      if (error.status === 401 || error.status === 403) {

        // Token was removed by api.js, let the component handle the state change

      }

      return rejectWithValue(error.body?.error || 'Ошибка загрузки сравнения');

    }

  }

);

export const saveCompareToServer = createAsyncThunk(

  'products/saveCompareToServer',

  async ({ userId, items }, { rejectWithValue }) => {

    try {

      console.log('Saving compare to server for user:', userId, 'items:', items);

      await updateCompare(userId, items);

      console.log('Successfully saved compare to server');

      return items;

    } catch (error) {

      console.error('Error saving compare to server:', error);

      // The token should have been removed by api.js, component will handle the auth state

      if (error.status === 401 || error.status === 403) {

        // Token was removed by api.js, let the component handle the state change

      }

      return rejectWithValue(error.body?.error || 'Ошибка сохранения сравнения');

    }

  }

);

const productsSlice = createSlice({

  name: 'products',

  initialState: {

    items: [],

    filteredItems: [],

    favorites: [],

    compare: [],

    status: 'idle',

    filters: {

      category: '',

      brand: '',

      minPrice: 0,

      maxPrice: 10000,

      search: ''

    }

  },

  reducers: {

    setCategoryFilter: (state, action) => {

      state.filters.category = action.payload;

      state.filteredItems = applyFilters(state.items, state.filters);

    },

    setBrandFilter: (state, action) => {

      state.filters.brand = action.payload;

      state.filteredItems = applyFilters(state.items, state.filters);

    },

    setPriceFilter: (state, action) => {

      state.filters.minPrice = action.payload.min;

      state.filters.maxPrice = action.payload.max;

      state.filteredItems = applyFilters(state.items, state.filters);

    },

    setSearchFilter: (state, action) => {

      state.filters.search = action.payload;

      state.filteredItems = applyFilters(state.items, state.filters);

    },

    resetFilters: (state) => {

      state.filters = {

        category: '',

        brand: '',

        minPrice: 0,

        maxPrice: 10000,

        search: ''

      };

      state.filteredItems = state.items;

    },

    addToFavorite: (state, action) => {

      console.log('User action: addToFavorite', action.payload);

      const existingItem = state.favorites.find(item => item.id === action.payload.id);

      if (!existingItem) {

        state.favorites.push(action.payload);

      }

    },

    removeFromFavorite: (state, action) => {

      console.log('User action: removeFromFavorite', action.payload);

      state.favorites = state.favorites.filter(item => item.id !== action.payload);

    },

    clearFavorites: (state) => {

      state.favorites = [];

    },

    addToCompare: (state, action) => {

      console.log('User action: addToCompare', action.payload);

      const existingItem = state.compare.find(item => item.id === action.payload.id);

      if (!existingItem && state.compare.length < 4) { // Limit to 4 items for comparison

        state.compare.push(action.payload);

      }

    },

    removeFromCompare: (state, action) => {

      console.log('User action: removeFromCompare', action.payload);

      state.compare = state.compare.filter(item => item.id !== action.payload);

    },

    clearCompare: (state) => {

      state.compare = [];

    },

    setFavorites: (state, action) => {

      state.favorites = action.payload;

    },

    setCompare: (state, action) => {

      state.compare = action.payload;

    }

  },

  extraReducers: builder => {

    builder

      .addCase(fetchProducts.pending, state => { state.status = 'loading'; })

      .addCase(fetchProducts.fulfilled, (state, action) => {

        state.items = action.payload;

        state.filteredItems = action.payload;

        state.status = 'succeeded';

      })

      .addCase(loadFavsFromServer.fulfilled, (state, action) => {

        state.favorites = action.payload;

      })

      .addCase(saveFavsToServer.fulfilled, (state, action) => {

        state.favorites = action.payload;

      })

      .addCase(loadCompareFromServer.fulfilled, (state, action) => {

        state.compare = action.payload;

      })

      .addCase(saveCompareToServer.fulfilled, (state, action) => {

        state.compare = action.payload;

      });

  },

});

// Вспомогательная функция для применения фильтров

const applyFilters = (items, filters) => {

  return items.filter(item => {

    // Фильтр по категории

    if (filters.category && item.category !== filters.category) {

      return false;

    }

    // Фильтр по бренду

    if (filters.brand && item.brand !== filters.brand) {

      return false;

    }

    // Фильтр по цене

    if (item.price < filters.minPrice || item.price > filters.maxPrice) {

      return false;

    }

    // Фильтр по поиску

    if (filters.search &&

        !item.title.toLowerCase().includes(filters.search.toLowerCase()) &&

        !item.animeSeries.toLowerCase().includes(filters.search.toLowerCase()) &&

        !item.character.toLowerCase().includes(filters.search.toLowerCase())) {

      return false;

    }

    return true;

  });

};

export const {

  setCategoryFilter,

  setBrandFilter,

  setPriceFilter,

  setSearchFilter,

  resetFilters,

  addToFavorite,

  removeFromFavorite,

  clearFavorites,

  addToCompare,

  removeFromCompare,

  clearCompare,

  setFavorites,

  setCompare

} = productsSlice.actions;

export default productsSlice.reducer;

Листинг 18 – ProductSlice.js

import React, { useState } from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { removeFromCart, updateQuantity, clearCart } from '../features/cart/cartSlice';

import { createOrder, fetchOrdersByUser } from '../store/orderSlice';

import { useNavigate } from 'react-router-dom';

import { selectUser } from '../store/authSlice';

import styles from '../styles/Cart.module.css';

function Cart() {

  const cartItems = useSelector(state => state.cart.items);

  const user = useSelector(selectUser);

  const dispatch = useDispatch();

  const navigate = useNavigate();

  const [orderForm, setOrderForm] = useState({

    firstName: user?.firstName || '',

    lastName: user?.lastName || '',

    phone: user?.phone || '',

    address: user?.address || '',

    email: user?.email || '',

    paymentMethod: 'card' // Default payment method

  });

  const [isOrdering, setIsOrdering] = useState(false);

  const [orderSuccess, setOrderSuccess] = useState(false);

  const handleRemove = (id) => {

    console.log('User clicked: Remove from Cart', id);

    dispatch(removeFromCart(id));

  };

  const handleQuantityChange = (id, newQuantity) => {

    console.log('User changed quantity for item', id, 'to', newQuantity);

    if (newQuantity <= 0) {

      dispatch(removeFromCart(id));

    } else {

      dispatch(updateQuantity({ id, quantity: newQuantity }));

    }

  };

  const getTotalPrice = () => {

    return cartItems.reduce((total, item) => total + (item.price \* item.quantity), 0);

  };

  const handleInputChange = (e) => {

    console.log('User changed input field', e.target.name, 'to', e.target.value);

    setOrderForm({

      ...orderForm,

      [e.target.name]: e.target.value

    });

  };

  const handleOrderSubmit = async (e) => {

    e.preventDefault();

    console.log('User submitted order form', orderForm);

    if (!user) {

      alert('Для оформления заказа необходимо войти в аккаунт');

      return;

    }

    setIsOrdering(true);

    try {

      const orderData = {

        userId: user.id,

        items: cartItems.map(item => ({

          id: item.id,

          title: item.title,

          quantity: item.quantity,

          price: item.price

        })),

        total: getTotalPrice(),

        customerInfo: {

          firstName: orderForm.firstName,

          lastName: orderForm.lastName,

          phone: orderForm.phone,

          address: orderForm.address,

          email: orderForm.email

        },

        paymentMethod: orderForm.paymentMethod,

        status: 'pending',

        createdAt: new Date().toISOString()

      };

      console.log('Submitting order:', orderData);

      await dispatch(createOrder(orderData));

      // Обновляем список заказов пользователя

      if (user) {

        await dispatch(fetchOrdersByUser(user.id));

      }

      setOrderSuccess(true);

      // Clear cart after successful order

      dispatch(clearCart());

      setTimeout(() => {

        navigate('/profile');

      }, 2000);

    } catch (error) {

      console.error('Ошибка оформления заказа:', error);

      alert('Ошибка при оформлении заказа. Пожалуйста, попробуйте еще раз.');

    } finally {

      setIsOrdering(false);

    }

  };

  if (orderSuccess) {

    return (

      <section className={styles.cart}>

        <h2>Заказ оформлен!</h2>

        <p>Ваш заказ успешно оформлен. Мы свяжемся с вами в ближайшее время.</p>

      </section>

    );

  }

  return (

    <section className={styles.cart}>

      <h2>Корзина</h2>

      {cartItems.length === 0 ? (

        <p>Ваша корзина пуста.</p>

      ) : (

        <>

          <ul className={styles.list}>

            {cartItems.map(item => (

              <li key={item.id} className={styles.item}>

                <img src={item.image} alt={item.title} className={styles.image} />

                <div className={styles.info}>

                  <h3>{item.title}</h3>

                  <p>{item.price} BYN</p>

                  <div className={styles.quantityControl}>

                    <button

                      onClick={() => handleQuantityChange(item.id, item.quantity - 1)}

                      className={styles.quantityBtn}

                    >

                      -

                    </button>

                    <span className={styles.quantity}>{item.quantity}</span>

                    <button

                      onClick={() => handleQuantityChange(item.id, item.quantity + 1)}

                      className={styles.quantityBtn}

                    >

                      +

                    </button>

                  </div>

                  <p className={styles.total}>Итого: {item.price \* item.quantity} BYN</p>

                </div>

                <button onClick={() => handleRemove(item.id)} className={styles.remove}>

                  Удалить

                </button>

              </li>

            ))}

          </ul>

          <div className={styles.totalSection}>

            <h3>Общая стоимость: {getTotalPrice()} BYN</h3>

            {/\* Order Form \*/}

            <div className={styles.orderForm}>

              <h3>Оформить заказ</h3>

              <form onSubmit={handleOrderSubmit}>

                <div className={styles.formGroup}>

                  <label htmlFor="firstName">Имя:</label>

                  <input

                    type="text"

                    id="firstName"

                    name="firstName"

                    value={orderForm.firstName}

                    onChange={handleInputChange}

                    required

                  />

                </div>

                <div className={styles.formGroup}>

                  <label htmlFor="lastName">Фамилия:</label>

                  <input

                    type="text"

                    id="lastName"

                    name="lastName"

                    value={orderForm.lastName}

                    onChange={handleInputChange}

                    required

                  />

                </div>

                <div className={styles.formGroup}>

                  <label htmlFor="phone">Телефон:</label>

                  <input

                    type="tel"

                    id="phone"

                    name="phone"

                    value={orderForm.phone}

                    onChange={handleInputChange}

                    required

                  />

                </div>

                <div className={styles.formGroup}>

                  <label htmlFor="address">Адрес:</label>

                  <input

                    type="text"

                    id="address"

                    name="address"

                    value={orderForm.address}

                    onChange={handleInputChange}

                    required

                  />

                </div>

                <div className={styles.formGroup}>

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

                  <input

                    type="email"

                    id="email"

                    name="email"

                    value={orderForm.email}

                    onChange={handleInputChange}

                    required

                  />

                </div>

                <div className={styles.formGroup}>

                  <label htmlFor="paymentMethod">Способ оплаты:</label>

                  <select

                    id="paymentMethod"

                    name="paymentMethod"

                    value={orderForm.paymentMethod}

                    onChange={handleInputChange}

                    required

                  >

                    <option value="card">Банковская карта</option>

                    <option value="cash">Наличные при получении</option>

                    <option value="online">Онлайн оплата</option>

                  </select>

                </div>

                <button

                  type="submit"

                  className={styles.orderBtn}

                  disabled={isOrdering}

                >

                  {isOrdering ? 'Оформление...' : 'Оформить заказ'}

                </button>

              </form>

            </div>

          </div>

        </>

      )}

    </section>

  );

}

export default Cart;

Листинг 18- Cart.Jsx

import React, { useEffect } from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { useSearchParams } from 'react-router-dom';

import { fetchProducts, setCategoryFilter, setBrandFilter, setPriceFilter, setSearchFilter } from '../features/products/productsSlice';

import { selectUser } from '../store/authSlice';

import ProductCard from '../components/ProductCard';

import Filters from '../components/Filters';

import styles from '../styles/Catalog.module.css';

function Catalog() {

  const dispatch = useDispatch();

  const user = useSelector(selectUser);

  const products = useSelector(state => state.products.filteredItems); // Используем отфильтрованные товары

  const status = useSelector(state => state.products.status);

  const [searchParams] = useSearchParams();

  useEffect(() => {

    if (status === 'idle') {

      dispatch(fetchProducts());

    }

  }, [dispatch, status]);

  // Применяем фильтры из URL при загрузке компонента

  useEffect(() => {

    const category = searchParams.get('category');

    const brand = searchParams.get('brand');

    const search = searchParams.get('search');

    const minPrice = searchParams.get('minPrice');

    const maxPrice = searchParams.get('maxPrice');

    if (category) {

      dispatch(setCategoryFilter(category));

    }

    if (brand) {

      dispatch(setBrandFilter(brand));

    }

    if (search) {

      dispatch(setSearchFilter(search));

    }

    if (minPrice || maxPrice) {

      dispatch(setPriceFilter({

        min: minPrice ? parseInt(minPrice) : undefined,

        max: maxPrice ? parseInt(maxPrice) : undefined

      }));

    }

  }, [dispatch, searchParams]);

  if (status === 'loading') {

    return <div>Загрузка товаров...</div>;

  }

  return (

    <section className={styles.catalog}>

      <h2>Каталог товаров</h2>

      <Filters />

      <div className={styles.productsGrid}>

        {products.map(p => (

          <ProductCard key={p.id} product={p} />

        ))}

      </div>

      {products.length === 0 && <p>Товары не найдены по заданным фильтрам</p>}

    </section>

  );

}

export default Catalog;

Листинг 19 – Catalog.jsx

import React, { useMemo } from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { removeFromCompare, clearCompare } from '../features/products/productsSlice';

import { selectUser } from '../store/authSlice';

import styles from '../styles/Compare.module.css';

function Compare() {

  const compareItems = useSelector(state => state.products.compare);

  const user = useSelector(selectUser);

  const dispatch = useDispatch();

  const handleRemove = (id) => {

    dispatch(removeFromCompare(id));

  };

  const clearAll = () => {

    dispatch(clearCompare());

  };

  // Group items by category

  const groupItemsByCategory = () => {

    const grouped = {};

    compareItems.forEach(item => {

      const category = item.category || 'Без категории';

      if (!grouped[category]) {

        grouped[category] = [];

      }

      grouped[category].push(item);

    });

    return grouped;

  };

  // Get common properties for items in the same category

  const getCommonProperties = (items) => {

    if (items.length === 0) return [];

    const allProps = new Set();

    items.forEach(item => {

      Object.keys(item).forEach(key => {

        if (typeof item[key] !== 'object' && key !== 'id' && key !== 'image' && key !== 'title') {

          allProps.add(key);

        }

      });

    });

    // Find properties that exist in all items

    const commonProps = [];

    for (const prop of allProps) {

      if (items.every(item => item.hasOwnProperty(prop))) {

        commonProps.push(prop);

      }

    };

    return commonProps;

  };

  const groupedItems = useMemo(() => {

    const grouped = {};

    compareItems.forEach(item => {

      const category = item.category || 'Без категории';

      if (!grouped[category]) {

        grouped[category] = [];

      }

      grouped[category].push(item);

    });

    return grouped;

  }, [compareItems]);

  const memoizedGetCommonProperties = useMemo(() => {

    return (items) => {

      if (items.length === 0) return [];

      const allProps = new Set();

      items.forEach(item => {

        Object.keys(item).forEach(key => {

          if (typeof item[key] !== 'object' && key !== 'id' && key !== 'image' && key !== 'title') {

            allProps.add(key);

          }

        });

      });

      // Find properties that exist in all items

      const commonProps = [];

      for (const prop of allProps) {

        if (items.every(item => item.hasOwnProperty(prop))) {

          commonProps.push(prop);

        }

      };

      return commonProps;

    };

  }, []);

  return (

    <div className={styles.container}>

      <div className={styles.header}>

        <h2>Сравнение товаров</h2>

        {compareItems.length > 0 && (

          <button onClick={clearAll} className={styles.clearBtn}>

            Очистить все

          </button>

        )}

      </div>

      {compareItems.length === 0 ? (

        <p>Нет товаров для сравнения.</p>

      ) : (

        Object.entries(groupedItems).map(([category, items]) => {

          const properties = memoizedGetCommonProperties(items);

          // Determine the number of products to apply appropriate CSS class

          const getTableClass = (count) => {

            if (count <= 2) return `${styles.compareTable} ${styles['two-products']}`;

            if (count === 3) return `${styles.compareTable} ${styles['three-products']}`;

            return `${styles.compareTable} ${styles['four-or-more-products']}`;

          };

          return (

            <div key={category} className={styles.categoryGroup}>

              <h3 className={styles.categoryTitle}>{category}</h3>

              <div className={getTableClass(items.length)}>

                {/\* Header row with product images and titles \*/}

                <div className={styles.headerRow}>

                  <div className={styles.propertyCell}>Свойство</div>

                  {items.map(item => (

                    <div key={item.id} className={styles.productCell}>

                      <img src={item.image} alt={item.title} className={styles.productImage} />

                      <h4>{item.title}</h4>

                      <button

                        onClick={() => handleRemove(item.id)}

                        className={styles.removeProductBtn}

                      >

                        Удалить

                      </button>

                    </div>

                  ))}

                </div>

                {/\* Property rows \*/}

                {properties.map(property => (

                  <div key={property} className={styles.dataRow}>

                    <div className={styles.propertyCell}>

                      <strong>{formatPropertyName(property)}</strong>

                    </div>

                    {items.map(item => (

                      <div key={`${item.id}-${property}`} className={styles.productCell}>

                        {item[property] || 'Не указано'}

                      </div>

                    ))}

                  </div>

                ))}

              </div>

            </div>

          );

        })

      )}

    </div>

  );

}

// Helper function to format property names

const formatPropertyName = (prop) => {

  const nameMap = {

    'price': 'Цена',

    'category': 'Категория',

    'animeSeries': 'Аниме',

    'character': 'Персонаж',

    'brand': 'Бренд',

    'material': 'Материал',

    'size': 'Размер',

    'ageRestriction': 'Возраст',

    'rating': 'Рейтинг',

    'description': 'Описание'

  };

  return nameMap[prop] || prop.charAt(0).toUpperCase() + prop.slice(1);

};

export default Compare;

Листинг 20 – Compare.jsx

import React from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { removeFromFavorite } from '../features/products/productsSlice';

import { addToCart } from '../features/cart/cartSlice';

import { selectUser } from '../store/authSlice';

import styles from '../styles/favourites.module.css';

function Favorites() {

  const favorites = useSelector(state => state.products.favorites);

  const user = useSelector(selectUser);

  const dispatch = useDispatch();

  const handleRemove = (id) => {

    dispatch(removeFromFavorite(id));

  };

  const handleAddToCart = (item) => {

    dispatch(addToCart(item));

    dispatch(removeFromFavorite(item.id));  // Remove from favorites when adding to cart

  };

  // Вспомогательная функция для получения понятного названия категории

  const getCategoryName = (category) => {

    const categoryNames = {

      'figurines': 'Фигурки',

      'posters': 'Плакаты',

      'books': 'Книги',

      'jewelry': 'Украшения',

      'clothing': 'Одежда',

      'accessories': 'Аксессуары',

      'cosplay': 'Косплей',

      'sweets': 'Сладости'

    };

    return categoryNames[category] || category;

  };

  return (

    <div className={styles.container}>

      <h2>Избранное</h2>

      {favorites.length === 0 ? (

        <p>У вас нет избранных товаров.</p>

      ) : (

        <div className={styles.favoritesGrid}>

          {favorites.map(item => (

            <div key={item.id} className={styles.favoriteItem}>

              <img src={item.image} alt={item.title} className={styles.image} />

              <div className={styles.info}>

                <h3 className={styles.title}>{item.title}</h3>

                <p className={styles.price}>{item.price} BYN</p>

                <p className={styles.category}>{getCategoryName(item.category)}</p>

                <p className={styles.animeSeries}>Аниме: {item.animeSeries}</p>

                <p className={styles.character}>Персонаж: {item.character}</p>

                <div className={styles.buttonGroup}>

                  <button

                    onClick={() => handleAddToCart(item)}

                    className={styles.addToCartBtn}

                  >

                    Добавить в корзину

                  </button>

                  <button

                    onClick={() => handleRemove(item.id)}

                    className={styles.removeBtn}

                  >

                    Удалить из избранного

                  </button>

                </div>

              </div>

            </div>

          ))}

        </div>

      )}

    </div>

  );

}

export default Favorites;

Листинг 21 – Favourites.jsx

import React from 'react';

import { Link } from 'react-router-dom';

import styles from '../styles/Home.module.css';

function Home() {

  // Category data for the cards

  const categories = [

    { id: 1, name: 'Фигурки', image: '/src/assets/categories/figures.jpg', path: '/catalog?category=figurines', description: 'Коллекционные фигурки персонажей' },

    { id: 2, name: 'Книги', image: '/src/assets/categories/books.jpg', path: '/catalog?category=books', description: 'Манга, ранобэ и книги' },

    { id: 3, name: 'Одежда', image: '/src/assets/categories/clothing.jpg', path: '/catalog?category=clothing', description: 'Одежда и аксессуары' },

    { id: 4, name: 'Плакаты', image: '/src/assets/categories/posters.jpg', path: '/catalog?category=posters', description: 'Постеры' },

  ];

  // News data

  const news = [

    { id: 1, title: 'Новый сезон аниме My Hero Academia', date: '2023-06-15', content: 'Объявлено о новом сезоне популярного аниме My Hero Academia с выходом в следующем году.' },

    { id: 2, title: 'Распродажа в честь лета', date: '2023-06-10', content: 'Скидки до 50% на коллекционные фигурки в честь летнего сезона.' },

    { id: 3, title: 'Новые поступления', date: '2023-06-05', content: 'В продаже появились эксклюзивные товары из аниме Demon Slayer.' },

  ];

  return (

    <section className={styles.home}>

      <div className={styles.hero}>

        <h1>Добро пожаловать в Anime Shop!</h1>

        <p>Широкий выбор товаров для фанатов аниме и манги</p>

      </div>

      <div className={styles.cta}>

        <Link to="/catalog" className={styles.ctaButton}>Перейти в каталог</Link>

      </div>

      <div className={styles.categories}>

        <h2>Популярные категории</h2>

        <div className={styles.categoryGrid}>

          {categories.map(category => (

            <Link to={category.path} key={category.id} className={styles.categoryCard}>

              <div className={styles.categoryImage}>

                <img src={category.image} alt={category.name} />

              </div>

              <div className={styles.categoryInfo}>

                <h3>{category.name}</h3>

                <p>{category.description}</p>

              </div>

            </Link>

          ))}

        </div>

      </div>

      <div className={styles.news}>

        <h2>Новости</h2>

        <div className={styles.newsList}>

          {news.map(item => (

            <div key={item.id} className={styles.newsItem}>

              <h3>{item.title}</h3>

              <p className={styles.date}>{item.date}</p>

              <p>{item.content}</p>

            </div>

          ))}

        </div>

      </div>

    </section>

  );

}

export default Home;

Листинг 22 – Home.jsx

import React from 'react';

import { useParams, useNavigate } from 'react-router-dom';

import { useSelector } from 'react-redux';

import { selectUser } from '../store/authSlice';

import styles from '../styles/OrderDetails.module.css';

function OrderDetails() {

  const { orderId } = useParams();

  const user = useSelector(selectUser);

  const navigate = useNavigate();

  // Получаем все заказы пользователя

  const allOrders = useSelector(state => {

    const orders = state.orders?.[user?.id] || [];

    console.log('OrderDetails: Все заказы пользователя из Redux', user?.id, ':', orders);

    return orders;

  });

  // Находим конкретный заказ по ID

  const order = allOrders.find(order => order.id === orderId);

  console.log('OrderDetails: Найденный заказ по ID', orderId, ':', order);

  if (!user) {

    return (

      <section className={styles.orderDetails}>

        <h2>Детали заказа</h2>

        <p>Пожалуйста, войдите в аккаунт для просмотра заказа.</p>

      </section>

    );

  }

  if (!order) {

    return (

      <section className={styles.orderDetails}>

        <h2>Детали заказа</h2>

        <p>Заказ не найден или у вас нет доступа к этому заказу.</p>

        <button onClick={() => navigate('/profile')}>Вернуться к профилю</button>

      </section>

    );

  }

  const getStatusLabel = (status) => {

    const statusLabels = {

      'pending': 'Ожидает обработки',

      'accepted': 'Принят продавцом',

      'processed': 'Обработан продавцом',

      'shipping': 'В доставке',

      'delivered': 'Заказ получен'

    };

    return statusLabels[status] || status;

  };

  const getPaymentMethodLabel = (method) => {

    const methodLabels = {

      'card': 'Банковская карта',

      'cash': 'Наличные при получении',

      'online': 'Онлайн оплата'

    };

    return methodLabels[method] || method;

  };

  return (

    <section className={styles.orderDetails}>

      <h2>Детали заказа №{order.id}</h2>

      <div className={styles.orderInfo}>

        <div className={styles.infoSection}>

          <h3>Информация о заказе</h3>

          <p><strong>Дата создания:</strong> {new Date(order.createdAt).toLocaleString()}</p>

          <p><strong>Статус:</strong> <span className={styles.status} data-status={order.status}>{getStatusLabel(order.status)}</span></p>

          <p><strong>Итоговая сумма:</strong> {order.total} BYN</p>

          <p><strong>Способ оплаты:</strong> {getPaymentMethodLabel(order.paymentMethod)}</p>

        </div>

        <div className={styles.customerSection}>

          <h3>Информация о покупателе</h3>

          <p><strong>Имя:</strong> {order.customerInfo?.firstName}</p>

          <p><strong>Фамилия:</strong> {order.customerInfo?.lastName}</p>

          <p><strong>Телефон:</strong> {order.customerInfo?.phone}</p>

          <p><strong>Email:</strong> {order.customerInfo?.email}</p>

          <p><strong>Адрес доставки:</strong> {order.customerInfo?.address}</p>

        </div>

      </div>

      <div className={styles.orderItems}>

        <h3>Товары в заказе</h3>

        <ul>

          {(order.items || []).map((item, index) => (

            <li key={index} className={styles.item}>

              <div className={styles.itemInfo}>

                <h4>{item.title || item.name}</h4>

                <p>Цена: {item.price} BYN</p>

                <p>Количество: {item.quantity || item.qty}</p>

                <p><strong>Итого: {(item.price \* (item.quantity || item.qty)).toFixed(2)} BYN</strong></p>

              </div>

            </li>

          ))}

        </ul>

      </div>

      <div className={styles.totalSection}>

        <h3>Итого</h3>

        <p><strong>Общая сумма:</strong> {order.total} BYN</p>

      </div>

      <button onClick={() => navigate('/profile')} className={styles.backButton}>

        Назад к профилю

      </button>

    </section>

  );

}

export default OrderDetails;

Листинг 23 – OrderDetails.jsx

import React, { useState, useEffect, useMemo, useCallback } from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { fetchOrdersByUser } from '../store/orderSlice';

import { useNavigate } from 'react-router-dom';

import styles from '../styles/Profile.module.css';

import { getUser } from '../utils/api.js';

import { selectUser, updateProfile } from '../store/authSlice';

function Profile() {

  const authUser = useSelector(selectUser);

  const dispatch = useDispatch();

  const navigate = useNavigate();

  const emptyOrders = useMemo(() => [], []);

  // Create a memoized selector to avoid recreating the selector function on each render

  const selectUserOrders = useCallback((state) => {

    console.log('Profile: Full orders state', state.orders);

    console.log('Profile: Auth user ID', authUser?.id);

    console.log('Profile: Checking if authUser exists', !!authUser);

    // Make sure the orders state exists before accessing it

    if (!state.orders) {

      console.warn('Profile: Orders state is undefined in Redux store');

      return emptyOrders;

    }

    const orders = state.orders?.[authUser?.id] || emptyOrders;

    console.log('Profile: Selector - заказы из состояния Redux для пользователя', authUser?.id, ':', orders);

    console.log('Profile: Length of orders', orders.length);

    return orders;

  }, [authUser?.id, emptyOrders]);

  const allOrders = useSelector(selectUserOrders);

  const [userData, setUserData] = useState({ lastName: '', firstName: '', phone: '', address: '' });

  const [orders, setOrders] = useState([]);

  const [saving, setSaving] = useState(false);

  const [loadingOrders, setLoadingOrders] = useState(false);

  const [error, setError] = useState(null);

  useEffect(() => {

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

    if (!token || !authUser) {

      // Если нет токена или пользователя, пользователь не авторизован

      console.log('Profile useEffect: Пользователь не авторизован или данные отсутствуют');

      return;

    }

    console.log('Profile useEffect: Загрузка данных профиля для пользователя', authUser);

    // Получаем информацию о пользователе с сервера

    const fetchUser = async () => {

      try {

        // Получаем актуальные данные пользователя с сервера

        try {

          console.log('Profile: Запрос данных пользователя с сервера для ID:', authUser.id);

          const userRes = await getUser(authUser.id);

          console.log('Profile: Ответ от сервера с данными пользователя', userRes);

          const freshUser = userRes.user;

          setUserData({

            lastName: freshUser.lastName || '',

            firstName: freshUser.firstName || '',

            phone: freshUser.phone || '',

            address: freshUser.address || ''

          });

          // Обновляем localStorage с актуальными данными

          localStorage.setItem('current\_user', JSON.stringify(freshUser));

          console.log('Profile: Данные пользователя обновлены в состоянии и localStorage');

        } catch (userErr) {

          console.error('Ошибка загрузки данных пользователя:', userErr);

          // If authentication failed (token removed), we expect the token to be removed in api.js

          if (userErr.status === 401 || userErr.status === 403) {

            // Token should have been removed by api.js, component will re-render with proper auth state

            return;

          }

          // Если не удалось получить данные с сервера, используем данные из Redux

          setUserData({

            lastName: authUser.lastName || '',

            firstName: authUser.firstName || '',

            phone: authUser.phone || '',

            address: authUser.address || ''

          });

        }

        // Загружаем историю заказов с сервера

        console.log('Profile: Начинаем загрузку заказов для пользователя', authUser.id);

        setLoadingOrders(true);

        try {

          console.log('Profile: Вызов dispatch(fetchOrdersByUser) для пользователя', authUser.id);

          const result = await dispatch(fetchOrdersByUser(authUser.id));

          console.log('Profile: Результат загрузки заказов', result);

          // Check if the action was rejected due to authentication

          if (fetchOrdersByUser.rejected.match(result) &&

              (result.payload?.includes('Invalid or expired token') ||

               result.payload?.includes('Access token required') ||

               result.error?.message?.includes('Invalid or expired token'))) {

            // Token should have been removed by api.js, component will re-render with proper auth state

            return;

          }

        } catch (ordersErr) {

          console.error('Ошибка загрузки заказов:', ordersErr);

          // If authentication failed (token removed), reload the page to reset state

          if (ordersErr.status === 401 || ordersErr.status === 403) {

            // Token should have been removed by api.js, component will re-render with proper auth state

            return;

          }

          // В случае ошибки, используем пустой массив заказов

          setOrders([]);

        } finally {

          setLoadingOrders(false);

        }

      } catch (err) {

        console.error('Ошибка загрузки данных профиля:', err);

        // If authentication failed (token removed), reload the page to reset state

        if (err.status === 401 || err.status === 403) {

          // Token should have been removed by api.js, component will re-render with proper auth state

          return;

        }

        setError('Ошибка загрузки данных профиля');

        setLoadingOrders(false);

      }

    };

    fetchUser();

  }, [authUser, dispatch]);

  const handleChange = (e) => {

    setUserData(prev => ({ ...prev, [e.target.name]: e.target.value }));

  };

  const handleSave = async () => {

    if (!authUser) return;

    setSaving(true);

    setError(null);

    const updatedUser = { id: authUser.id, ...userData };

    try {

      // Dispatch the updateProfile thunk

      const result = await dispatch(updateProfile(updatedUser));

      if (updateProfile.fulfilled.match(result)) {

        const serverUser = result.payload;

        // Update localStorage with server response

        localStorage.setItem('current\_user', JSON.stringify(serverUser));

        setSaving(false);

        alert('Данные сохранены');

      } else {

        setSaving(false);

        setError(result.payload || 'Ошибка при сохранении на сервере');

        alert('Ошибка при сохранении на сервере');

      }

    } catch (err) {

      setSaving(false);

      setError('Ошибка при сохранении на сервере');

      alert('Ошибка при сохранении на сервере');

    }

  };

  const handleLogout = () => {

    localStorage.removeItem('auth\_token');

    localStorage.removeItem('current\_user');

    // Перенаправляем на главную страницу

    window.location.href = '/';

  };

  const getStatusLabel = (status) => {

    const statusLabels = {

      'pending': 'Ожидает обработки',

      'accepted': 'Принят продавцом',

      'processed': 'Обработан продавцом',

      'shipping': 'В доставке',

      'delivered': 'Заказ получен'

    };

    return statusLabels[status] || status;

  };

  const getPaymentMethodLabel = (method) => {

    const methodLabels = {

      'card': 'Банковская карта',

      'cash': 'Наличные при получении',

      'online': 'Онлайн оплата'

    };

    return methodLabels[method] || method;

  };

  if (!authUser) {

    return (

      <section className={styles.profile}>

        <h2>Личный кабинет</h2>

        <p>Пожалуйста, войдите в аккаунт, чтобы просматривать профиль.</p>

      </section>

    );

  }

  return (

    <section className={styles.profile}>

      <h2>Личный кабинет</h2>

      <div className={styles.card}>

        <div className={styles.left}>

          <label>Фамилия

            <input name="lastName" value={userData.lastName} onChange={handleChange} />

          </label>

          <label>Имя

            <input name="firstName" value={userData.firstName} onChange={handleChange} />

          </label>

          <label>Телефон

            <input name="phone" value={userData.phone} onChange={handleChange} />

          </label>

          <label>Адрес

            <input name="address" value={userData.address} onChange={handleChange} />

          </label>

          <div className={styles.buttons}>

            <button onClick={handleSave} disabled={saving}>

              {saving ? 'Сохраняем...' : '💾 Сохранить'}

            </button>

            <button onClick={handleLogout}>🚪 Выйти</button>

          </div>

          {error && <p className={styles.error}>{error}</p>}

        </div>

        <div className={styles.right}>

          <p><strong>Email:</strong> {authUser.email}</p>

          <p><strong>Роль:</strong> {authUser.role}</p>

          <p><strong>Зарегистрирован:</strong> {new Date(authUser.createdAt).toLocaleDateString()}</p>

        </div>

      </div>

      <div className={styles.orders}>

        <h3>🧾 История заказов</h3>

        {loadingOrders ? (

          <p>Загрузка...</p>

        ) : allOrders.length === 0 ? (

          <p>У вас пока нет заказов.</p>

        ) : (

          allOrders.map(order => (

            <div

              key={order.id}

              className={styles.orderCard}

              onClick={() => navigate(`/order/${order.id}`)}

              style={{ cursor: 'pointer' }}

            >

              <div className={styles.orderHeader}>

                <p><strong>Заказ №:</strong> {order.id}</p>

                <p><strong>Дата:</strong> {new Date(order.createdAt).toLocaleString()}</p>

              </div>

              <ul>

                {(order.items || []).map((item, i) => (

                  <li key={i}>{item.title || item.name} — {item.quantity || item.qty} шт. × {item.price} BYN</li>

                ))}

              </ul>

              <p><strong>Итого:</strong> {order.total} BYN</p>

              <p><strong>Способ оплаты:</strong> {getPaymentMethodLabel(order.paymentMethod)}</p>

              <p><strong>Статус заказа:</strong> <span className={styles.orderStatus} data-status={order.status}>{getStatusLabel(order.status)}</span></p>

            </div>

          ))

        )}

      </div>

    </section>

  );

}

export default Profile;

Листинг 24 – Profile.jsx

import { createSlice, createAsyncThunk, createSelector } from '@reduxjs/toolkit';

import { register, login as loginApi, getUser, updateUser } from '../utils/api';

const initialState = {

  user: null,

  profile: null,

  status: 'idle',

  error: null

};

export const registerUser = createAsyncThunk(

  'auth/register',

  async (formData, { rejectWithValue }) => {

    try {

      const response = await register(formData);

      if (response.token) {

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

      }

      return response.user;

    } catch (error) {

      return rejectWithValue(error.body?.error || 'Ошибка регистрации');

    }

  }

);

export const login = createAsyncThunk(

  'auth/login',

  async (formData, { rejectWithValue }) => {

    try {

      const response = await loginApi(formData);

      if (response.token) {

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

      }

      return response.user;

    } catch (error) {

      return rejectWithValue(error.body?.error || 'Ошибка входа');

    }

  }

);

export const fetchUserProfile = createAsyncThunk(

  'auth/fetchProfile',

  async (userId, { rejectWithValue }) => {

    try {

      const response = await getUser(userId);

      return response.user;

    } catch (error) {

      // The token should have been removed by api.js, component will handle the auth state

      if (error.status === 401 || error.status === 403) {

        // Token was removed by api.js, let the component handle the state change

      }

      return rejectWithValue(error.body?.error || 'Ошибка получения профиля');

    }

  }

);

export const updateProfile = createAsyncThunk(

  'auth/updateProfile',

  async (userData, { rejectWithValue }) => {

    try {

      const { id, ...updateData } = userData;

      const response = await updateUser(id, updateData);

      return response.user;

    } catch (error) {

      // The token should have been removed by api.js, component will handle the auth state

      if (error.status === 401 || error.status === 403) {

        // Token was removed by api.js, let the component handle the state change

      }

      return rejectWithValue(error.body?.error || 'Ошибка обновления профиля');

    }

  }

);

const authSlice = createSlice({

  name: 'auth',

  initialState,

  reducers: {

    logout(state) {

      state.user = null;

      state.profile = null;

      state.status = 'idle';

      state.error = null;

      localStorage.removeItem('auth\_token');

      localStorage.removeItem('current\_user');

    },

    clearError(state) {

      state.error = null;

    },

    setUser(state, action) {

      state.user = action.payload;

    }

  },

  extraReducers: (builder) => {

    builder

      .addCase(registerUser.pending, (state) => {

        state.status = 'loading';

        state.error = null;

      })

      .addCase(registerUser.fulfilled, (state, action) => {

        state.status = 'succeeded';

        state.user = action.payload;

      })

      .addCase(registerUser.rejected, (state, action) => {

        state.status = 'failed';

        state.error = action.payload;

      })

      .addCase(login.pending, (state) => {

        state.status = 'loading';

        state.error = null;

      })

      .addCase(login.fulfilled, (state, action) => {

        state.status = 'succeeded';

        state.user = action.payload;

      })

      .addCase(login.rejected, (state, action) => {

        state.status = 'failed';

        state.error = action.payload;

      })

      .addCase(fetchUserProfile.fulfilled, (state, action) => {

        state.profile = action.payload;

      })

      .addCase(updateProfile.pending, (state) => {

        state.status = 'loading';

        state.error = null;

      })

      .addCase(updateProfile.fulfilled, (state, action) => {

        state.status = 'succeeded';

        state.user = action.payload;

        state.profile = action.payload;

      })

      .addCase(updateProfile.rejected, (state, action) => {

        state.status = 'failed';

        state.error = action.payload;

      });

  }

});

// --- Selectors ---

export const selectUser = (state) => state.auth.user;

export const selectProfile = (state) => state.auth.profile;

export const selectStatus = (state) => state.auth.status;

export const selectError = (state) => state.auth.error;

// Мемоизированный селектор для избежания лишних ререндеров

export const selectAuthState = createSelector(

  [selectUser, selectProfile, selectStatus, selectError],

  (user, profile, status, error) => ({

    user,

    profile,

    status,

    error

  })

);

export const { logout, clearError, setUser } = authSlice.actions;

export default authSlice.reducer;

Листинг 25 – authSlice.js

import { configureStore } from '@reduxjs/toolkit';

import { persistStore, persistReducer } from 'redux-persist';

import storage from 'redux-persist/lib/storage';

import authReducer from './authSlice';

import cartReducer from '../features/cart/cartSlice';

import productsReducer from '../features/products/productsSlice';

import ordersReducer from './orderSlice';

const authPersistConfig = {

  key: 'auth',

  storage,

};

const store = configureStore({

  reducer: {

    auth: persistReducer(authPersistConfig, authReducer),

    cart: cartReducer,

    products: productsReducer,

    orders: ordersReducer,

  },

  middleware: (getDefaultMiddleware) =>

    getDefaultMiddleware({

      serializableCheck: false,

    }),

});

export const persistor = persistStore(store);

export default store;

Листинг 26 – index.js

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

import { createOrder as createOrderApi, getOrdersByUser as getOrdersByUserApi } from '../utils/api';

// Async thunks for server sync

export const createOrder = createAsyncThunk(

  'orders/createOrder',

  async (orderData, { rejectWithValue }) => {

    try {

      const response = await createOrderApi(orderData);

      return response.order;

    } catch (error) {

      return rejectWithValue(error.body?.error || 'Ошибка создания заказа');

    }

  }

);

export const fetchOrdersByUser = createAsyncThunk(

  'orders/fetchByUser',

  async (userId, { rejectWithValue }) => {

    console.log('fetchOrdersByUser action вызвана для userId:', userId);

    try {

      const response = await getOrdersByUserApi(userId);

      console.log('Ответ от сервера для заказов пользователя:', userId, response);

      return { userId, orders: response.orders };

    } catch (error) {

      console.error('Ошибка загрузки заказов для пользователя:', userId, error);

      return rejectWithValue(error.body?.error || 'Ошибка загрузки заказов');

    }

  }

);

const ordersSlice = createSlice({

  name: 'orders',

  initialState: {},

  reducers: {

    setOrdersForUser(state, action) {

      const { userId, orders } = action.payload;

      state[userId] = orders;

    },

    clearOrders(state, action) {

      const { userId } = action.payload;

      delete state[userId];

    }

  },

  extraReducers: (builder) => {

    builder

      .addCase(createOrder.fulfilled, (state, action) => {

        const order = action.payload;

        console.log('createOrder.fulfilled: Новый заказ создан', order);

        state[order.userId] = state[order.userId] || [];

        state[order.userId].push(order);

        console.log('createOrder.fulfilled: Обновленное состояние заказов для пользователя', order.userId, state[order.userId]);

      })

      .addCase(fetchOrdersByUser.fulfilled, (state, action) => {

        const { userId, orders } = action.payload;

        console.log('fetchOrdersByUser.fulfilled: Загружено заказов для пользователя', userId, ':', orders?.length || 0);

        console.log('fetchOrdersByUser.fulfilled: Данные заказов', orders);

        state[userId] = Array.isArray(orders) ? [...orders] : [];

        console.log('fetchOrdersByUser.fulfilled: Обновленное состояние заказов для пользователя', userId, state[userId]);

        console.log('fetchOrdersByUser.fulfilled: Полное состояние после обновления', state);

      })

      .addCase(fetchOrdersByUser.rejected, (state, action) => {

        console.error('fetchOrdersByUser.rejected: Ошибка загрузки заказов', action.error.message);

        // Optionally handle authentication errors here as well

        if (action.payload && (action.payload.includes('Invalid or expired token') || action.payload.includes('Access token required'))) {

          // The token should have been removed by api.js, component will handle the auth state

          console.error('Authentication error detected in orderSlice');

          // Token was removed by api.js, let the component handle the state change

        }

      });

  }

});

export const { setOrdersForUser, clearOrders } = ordersSlice.actions;

export default ordersSlice.reducer;

Листинг 27 – orderSlice.js

const API\_ROOT = 'http://localhost:3001/api';

// Функция для добавления токена авторизации к запросу

function getAuthHeaders() {

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

  const headers = { 'Content-Type': 'application/json' };

  if (token) {

    headers['Authorization'] = `Bearer ${token}`;

  }

  return headers;

}

async function apiFetch(path, opts = {}) {

  const headers = opts.headers || getAuthHeaders();

  console.log('API: Выполняем запрос к', `${API\_ROOT}${path}`, 'с опциями:', opts);

  const res = await fetch(`${API\_ROOT}${path}`, {

    headers,

    ...opts

  });

  console.log('API: Получен ответ с статусом', res.status, 'и заголовками:', res.headers);

  const text = await res.text();

  console.log('API: Тело ответа как текст:', text);

  try {

    // Проверяем, что текст ответа существует и содержит валидный JSON

    const json = text && text.trim() ? JSON.parse(text) : {};

    console.log('API: Разобранный JSON ответа:', json);

    // If the response status is 401 or 403 (unauthorized/forbidden),

    // it likely means the token is invalid or expired

    if (res.status === 401 || res.status === 403) {

      // Remove the invalid token from localStorage

      localStorage.removeItem('auth\_token');

      localStorage.removeItem('current\_user');

      // Optionally, redirect user to login page or clear user data

      console.warn('API: Токен недействителен или истек, удаляем из localStorage');

    }

    if (!res.ok) throw { status: res.status, body: json };

    return json;

  } catch (err) {

    console.log('API: Ошибка парсинга ответа или статус не ok:', err);

    if (err && err.status) {

      // If the error is due to authentication, remove the token

      if (err.status === 401 || err.status === 403) {

        localStorage.removeItem('auth\_token');

        localStorage.removeItem('current\_user');

        console.warn('API: Токен недействителен или истек, удаляем из localStorage');

      }

      throw err;

    }

    // Если ошибка парсинга JSON, но статус успешный, возвращаем пустой объект

    if (res.ok) {

      console.warn(`Предупреждение: Невозможно распарсить JSON-ответ от ${path}, возвращаем пустой объект`);

      return {};

    }

    throw { status: res.status, message: 'Invalid JSON response' };

  }

}

export async function register(data) {

  return apiFetch('/register', { method: 'POST', body: JSON.stringify(data) });

}

export async function login(data) {

  return apiFetch('/login', { method: 'POST', body: JSON.stringify(data) });

}

export async function getUser(id) {

  return apiFetch(`/users/${id}`, { method: 'GET' });

}

export async function updateUser(id, data) {

  return apiFetch(`/users/${id}`, { method: 'PUT', body: JSON.stringify(data) });

}

// Orders

export async function createOrder(order) {

  return apiFetch('/orders', { method: 'POST', body: JSON.stringify(order) });

}

export async function getOrdersByUser(userId) {

  console.log('API: Запрос заказов пользователя с ID:', userId);

  const response = await apiFetch(`/orders/${userId}`, { method: 'GET' });

  console.log('API: Ответ на запрос заказов пользователя', userId, ':', response);

  // Проверяем формат ответа и возвращаем объект с полем orders, как ожидается в orderSlice

  if (Array.isArray(response)) {

    // Если сервер вернул массив (старый формат), оборачиваем в объект

    console.log('API: Сервер вернул массив, оборачиваем в объект с полем orders');

    return { orders: response };

  } else if (response.orders) {

    // Если сервер вернул объект с полем orders (новый формат)

    console.log('API: Сервер вернул объект с полем orders');

    return response;

  } else {

    // В других случаях возвращаем как есть

    console.log('API: Неожиданный формат ответа от сервера, возвращаем как есть');

    return response;

  }

}

// Cart

export async function getCart(userId) {

  return apiFetch(`/cart/${userId}`, { method: 'GET' });

}

export async function updateCart(userId, items) {

  return apiFetch(`/cart/${userId}`, { method: 'PUT', body: JSON.stringify({ items }) });

}

// Favs

export async function getFavs(userId) {

  return apiFetch(`/favs/${userId}`, { method: 'GET' });

}

export async function updateFavs(userId, items) {

  return apiFetch(`/favs/${userId}`, { method: 'PUT', body: JSON.stringify({ items }) });

}

// Compare

export async function getCompare(userId) {

  return apiFetch(`/compare/${userId}`, { method: 'GET' });

}

export async function updateCompare(userId, items) {

  return apiFetch(`/compare/${userId}`, { method: 'PUT', body: JSON.stringify({ items }) });

}

Листинг 28 – api.js

import { logout } from '../store/authSlice';

// Middleware для проверки аутентификации

export const authMiddleware = (store) => (next) => (action) => {

  // Проверяем, является ли действие, связанным с аутентификацией

  if (action.type.includes('auth/')) {

    // Проверяем токен при определенных действиях

    if (action.type === 'auth/login/fulfilled' || action.type === 'auth/register/fulfilled') {

      // После успешного входа или регистрации сохраняем токен в localStorage

      if (action.payload && action.payload.token) {

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

      }

    } else if (action.type === 'auth/logout') {

      // При выходе удаляем токен

      localStorage.removeItem('auth\_token');

      localStorage.removeItem('current\_user');

    }

  }

  return next(action);

};

// Функция для проверки токена при загрузке приложения

export const checkAuthTokenOnLoad = () => {

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

  // Если токен есть, но состояние Redux не отражает это,

  // возможно, приложение было перезагружено и нужно обновить состояние

  if (token) {

    // Токен есть, но Redux может не знать о пользователе

    // В этом случае мы можем попытаться восстановить состояние пользователя

    const user = localStorage.getItem('current\_user');

    if (user) {

      try {

        return JSON.parse(user);

      } catch (e) {

        console.warn('Ошибка при разборе данных пользователя из localStorage');

        localStorage.removeItem('current\_user');

      }

    }

  }

  return null;

};

Листинг 29 – authMiddleware.js

import usersSeed from '../../data/users.json';

const LS\_USERS = 'app\_users';

function ensureInit() {

  if (!localStorage.getItem(LS\_USERS)) {

    localStorage.setItem(LS\_USERS, JSON.stringify(usersSeed || []));

  }

}

export function getAllUsers() {

  ensureInit();

  try {

    return JSON.parse(localStorage.getItem(LS\_USERS) || '[]');

  } catch {

    return [];

  }

}

export function saveAllUsers(users) {

  localStorage.setItem(LS\_USERS, JSON.stringify(users));

}

export function getUserByEmail(email) {

  const users = getAllUsers();

  return users.find(u => u.email === email) || null;

}

export function getUserById(id) {

  const users = getAllUsers();

  return users.find(u => u.id === id) || null;

}

export function addUser(user) {

  const users = getAllUsers();

  users.push(user);

  saveAllUsers(users);

}

export function updateUser(id, patch) {

  const users = getAllUsers();

  const idx = users.findIndex(u => u.id === id);

  if (idx === -1) return null;

  users[idx] = { ...users[idx], ...patch };

  saveAllUsers(users);

  return users[idx];

}

Листинг 30 – dataService.js

import { useEffect, useRef, useCallback } from 'react';

import { useSelector, useDispatch } from 'react-redux';

import { saveCartToServer } from '../features/cart/cartSlice';

import { saveFavsToServer, saveCompareToServer } from '../features/products/productsSlice';

import { selectUser } from '../store/authSlice';

// Custom hook to sync Redux state with server when user is authenticated

export const useSyncWithServer = () => {

  const dispatch = useDispatch();

  const user = useSelector(selectUser);

  // Use refs to track current values and previous values for comparison

  const userRef = useRef();

  const cartItemsRef = useRef();

  const favoritesRef = useRef();

  const compareRef = useRef();

  // Use refs to track previous values for comparison

  const prevUser = useRef();

  const prevCartItems = useRef();

  const prevFavorites = useRef();

  const prevCompare = useRef();

  // Track if we're currently syncing to prevent recursive updates

  const isSyncing = useRef({

    cart: false,

    favs: false,

    compare: false

  });

  // Get current values and compare with previous values

  const currentCartItems = useSelector(state => state.cart.items);

  const currentFavorites = useSelector(state => state.products.favorites);

  const currentCompare = useSelector(state => state.products.compare);

  // Check if values have changed before updating refs

  useEffect(() => {

    userRef.current = user;

    // Only update refs if values have actually changed

    if (JSON.stringify(currentCartItems) !== JSON.stringify(cartItemsRef.current)) {

      cartItemsRef.current = currentCartItems;

    }

    if (JSON.stringify(currentFavorites) !== JSON.stringify(favoritesRef.current)) {

      favoritesRef.current = currentFavorites;

    }

    if (JSON.stringify(currentCompare) !== JSON.stringify(compareRef.current)) {

      compareRef.current = currentCompare;

    }

  }, [user, currentCartItems, currentFavorites, currentCompare]);

  useEffect(() => {

    const userId = userRef.current?.id;

    if (!userId) return;

    // Check if cart items have changed

    const currentCartItemsStr = JSON.stringify(cartItemsRef.current);

    if (!prevCartItems.current || currentCartItemsStr !== prevCartItems.current) {

      if (!isSyncing.current.cart) {

        console.log('Cart changed, syncing to server:', cartItemsRef.current);

        isSyncing.current.cart = true;

        dispatch(saveCartToServer({ userId, items: cartItemsRef.current }))

          .finally(() => {

            isSyncing.current.cart = false;

          });

      }

      prevCartItems.current = currentCartItemsStr;

    }

    // Check if favorites have changed

    const currentFavoritesStr = JSON.stringify(favoritesRef.current);

    if (!prevFavorites.current || currentFavoritesStr !== prevFavorites.current) {

      if (!isSyncing.current.favs) {

        console.log('Favorites changed, syncing to server:', favoritesRef.current);

        isSyncing.current.favs = true;

        dispatch(saveFavsToServer({ userId, items: favoritesRef.current }))

          .finally(() => {

            isSyncing.current.favs = false;

          });

      }

      prevFavorites.current = currentFavoritesStr;

    }

    // Check if compare items have changed

    const currentCompareStr = JSON.stringify(compareRef.current);

    if (!prevCompare.current || currentCompareStr !== prevCompare.current) {

      if (!isSyncing.current.compare) {

        console.log('Compare items changed, syncing to server:', compareRef.current);

        isSyncing.current.compare = true;

        dispatch(saveCompareToServer({ userId, items: compareRef.current }))

          .finally(() => {

            isSyncing.current.compare = false;

          });

      }

      prevCompare.current = currentCompareStr;

    }

  }, [dispatch]); // Only depend on dispatch to prevent constant re-runs due to state changes

};

Листинг 31 – syncUtils.js

import React from 'react';

import { createRoot } from 'react-dom/client';

import { Provider } from 'react-redux';

import { BrowserRouter } from 'react-router-dom';

import { PersistGate } from 'redux-persist/integration/react';

import store, { persistor } from './store';

import App from './App';

import './styles/global.css';

const container = document.getElementById('root');

const root = createRoot(container);

root.render(

  <React.StrictMode>

    <Provider store={store}>

      <PersistGate loading={null} persistor={persistor}>

        <BrowserRouter>

          <App />

        </BrowserRouter>

      </PersistGate>

    </Provider>

  </React.StrictMode>

);

Листинг 32 – index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './app/App';

import { BrowserRouter } from 'react-router-dom';

import { Provider } from 'react-redux';

import { PersistGate } from 'redux-persist/integration/react';

import store, { persistor } from './app/store';

import { NotificationProvider } from './contexts/NotificationContext';

import './index.css'; // Import global styles

ReactDOM.createRoot(document.getElementById('root')).render(

  <React.StrictMode>

    <Provider store={store}>

      <PersistGate loading={null} persistor={persistor}>

        <BrowserRouter>

          <NotificationProvider>

            <App />

          </NotificationProvider>

        </BrowserRouter>

      </PersistGate>

    </Provider>

  </React.StrictMode>

);

Листинг 33 – main.jsx

// Исправленный обработчик получения заказов пользователя

// Этот код должен заменить проблемный участок в server.js строке 304

import { Router } from ‘express’;

import fs from ‘fs/promises’;

import path from ‘path’;

const router = Router();

// Функция для безопасного разбора JSON

function safeJsonParse(data, defaultValue = null) {

  if (!data || data.length === 0) {

    return defaultValue;

  }

  try {

    return JSON.parse(data);

  } catch (error) {

    console.error(‘Ошибка парсинга JSON:’, error);

    return defaultValue;

  }

}

// Маршрут для получения заказов пользователя

router.get(‘/orders/:userId’, async (req, res) => {

  try {

    const userId = req.params.userId;

    // Проверяем, что userId предоставлен

    if (!userId) {

      return res.status(400).json({ error: ‘ID пользователя обязателен’ });

    }

    // Путь к файлу с заказами

    const ordersFilePath = path.join(process.cwd(), ‘data’, ‘orders.json’);

    let orders = [];

    // Проверяем существование файла

    try {

      const data = await fs.readFile(ordersFilePath, ‘utf8’);

      orders = safeJsonParse(data, []) || [];

    } catch (readError) {

      if (readError.code === ‘ENOENT’) {

        // Файл не существует, возвращаем пустой массив

        orders = [];

      } else {

        throw readError;

      }

    }

    // Фильтруем заказы по ID пользователя

    const userOrders = orders.filter(order => order.userId === userId);

    // Возвращаем заказы пользователя

    res.json(userOrders);

  } catch (error) {

    console.error(‘Ошибка получения заказов:’, error);

    res.status(500).json({ error: ‘Ошибка сервера при получении заказов’ });

  }

});

export default router;

Листинг 34 – serverorderfix.js

### ПРИЛОЖЕНИЕ Г. Листинг css документов

.cart {

  padding: 30px 20px;

  max-width: 1200px;

  margin: 0 auto;

  background: linear-gradient(135deg, #ffffff, #f8f9fa);

  min-height: calc(100vh - 200px);

  border-radius: 15px;

  box-shadow: 0 10px 30px rgba(0,0,0,0.08);

  margin-top: 20px;

  margin-bottom: 20px;

  position: relative;

  overflow: hidden;

  animation: fadeInUp 0.8s ease-out;

}

.cart::before {

  content: '';

  position: absolute;

  top: 0;

  left: 0;

  right: 0;

  height: 5px;

  background: linear-gradient(90deg, #e91e63, #667eea);

  animation: gradientShift 3s infinite alternate;

}

@keyframes gradientShift {

  0% { background: linear-gradient(90deg, #e91e63, #667eea); }

  50% { background: linear-gradient(90deg, #667eea, #e91e63); }

  100% { background: linear-gradient(90deg, #e91e63, #667eea); }

}

.cart h2 {

  text-align: center;

  font-size: 2.5rem;

  margin-bottom: 30px;

  color: #2c3e50;

  text-shadow: 2px 2px 4px rgba(0,0,0,0.1);

  background: linear-gradient(135deg, #e91e63, #667eea);

  -webkit-background-clip: text;

  -webkit-text-fill-color: transparent;

  background-clip: text;

  position: relative;

  display: inline-block;

}

.cart h2::after {

  content: '';

  position: absolute;

  bottom: -10px;

  left: 50%;

  transform: translateX(-50%);

  width: 80px;

  height: 3px;

  background: linear-gradient(90deg, #e91e63, #667eea);

  border-radius: 2px;

}

.list {

  list-style: none;

  padding: 0;

  margin: 0;

}

.item {

  display: flex;

  align-items: center;

  margin-bottom: 20px;

  border-radius: 12px;

  padding: 15px;

  background: white;

  box-shadow: 0 4px 10px rgba(0,0,0,0.08);

  transition: all 0.3s ease;

  border-left: 4px solid #e91e63;

  animation: fadeInUp 0.5s ease-out;

}

.item:hover {

  transform: translateY(-3px);

  box-shadow: 0 8px 20px rgba(0,0,0,0.12);

}

.image {

  width: 100px;

  height: 100px;

  object-fit: cover;

  margin-right: 20px;

  border-radius: 8px;

  box-shadow: 0 4px 8px rgba(0,0,0,0.1);

}

.info {

  flex: 1;

}

.quantityControl {

  display: flex;

  align-items: center;

  margin: 12px 0;

  justify-content: flex-start;

}

.quantityBtn {

  background: linear-gradient(135deg, #6c757d, #5a6268);

  color: white;

  border: none;

  width: 36px;

  height: 36px;

  border-radius: 50%;

  cursor: pointer;

  font-size: 18px;

  display: flex;

  align-items: center;

  justify-content: center;

  transition: all 0.3s ease;

  box-shadow: 0 2px 5px rgba(0,0,0,0.1);

}

.quantityBtn:hover {

  background: linear-gradient(135deg, #5a6268, #495057);

  transform: scale(1.1);

}

.quantity {

  font-weight: bold;

  min-width: 40px;

  text-align: center;

  margin: 0 12px;

  font-size: 18px;

  color: #e91e63;

}

.total {

  font-weight: bold;

  color: #e91e63;

  margin-top: 8px;

  font-size: 18px;

  text-shadow: 1px 1px 2px rgba(0,0,0,0.05);

}

.remove {

  background: linear-gradient(135deg, #e91e63, #c2185b);

  color: white;

  border: none;

  padding: 10px 16px;

  cursor: pointer;

  border-radius: 25px;

  transition: all 0.3s ease;

  box-shadow: 0 4px 8px rgba(233, 30, 99, 0.3);

  font-weight: 500;

}

.remove:hover {

  background: linear-gradient(135deg, #c2185b, #ad1457);

  transform: translateY(-2px);

  box-shadow: 0 6px 12px rgba(233, 30, 99, 0.4);

}

.totalSection {

  margin-top: 30px;

  padding: 25px;

  background: linear-gradient(135deg, #f8f9fa, #e9ecef);

  border-radius: 15px;

  text-align: right;

  box-shadow: 0 4px 15px rgba(0,0,0,0.08);

  border: 1px solid #dee2e6;

}

.totalSection h3 {

  font-size: 1.8rem;

  color: #e91e63;

  margin-top: 0;

  margin-bottom: 20px;

  text-shadow: 1px 1px 2px rgba(0,0,0,0.05);

}

.orderForm {

  margin-top: 30px;

  padding: 25px;

  background: white;

  border-radius: 15px;

  box-shadow: 0 6px 20px rgba(0,0,0,0.08);

  border-left: 5px solid #28a745;

  animation: fadeInUp 0.8s ease-out 0.2s both;

}

.orderForm h3 {

  margin-top: 0;

  margin-bottom: 20px;

  color: #333;

  font-size: 1.6rem;

  text-align: center;

  background: linear-gradient(135deg, #28a745, #218838);

  -webkit-background-clip: text;

  -webkit-text-fill-color: transparent;

  background-clip: text;

}

.formGroup {

  margin-bottom: 20px;

}

.formGroup label {

  display: block;

  margin-bottom: 8px;

  font-weight: bold;

  color: #555;

  font-size: 14px;

  text-transform: uppercase;

  letter-spacing: 0.5px;

}

.formGroup input,

.formGroup select {

  width: 100%;

  padding: 12px 16px;

  border: 2px solid #e0e0e0;

  border-radius: 8px;

  font-size: 16px;

  box-sizing: border-box;

  transition: all 0.3s ease;

  background-color: #f8f9fa;

}

.formGroup input:focus,

.formGroup select:focus {

  outline: none;

  border-color: #e91e63;

  box-shadow: 0 0 0 3px rgba(233, 30, 99, 0.2);

  background-color: white;

}

.orderBtn {

  background: linear-gradient(135deg, #28a745, #218838);

  color: white;

  border: none;

  padding: 14px 25px;

  font-size: 18px;

  border-radius: 30px;

  cursor: pointer;

  width: 100%;

  font-weight: bold;

  transition: all 0.3s ease;

  box-shadow: 0 6px 15px rgba(40, 167, 69, 0.4);

  margin-top: 10px;

}

.orderBtn:hover {

  background: linear-gradient(135deg, #218838, #1e7e34);

  transform: translateY(-3px);

  box-shadow: 0 8px 20px rgba(40, 167, 69, 0.5);

}

.orderBtn:disabled {

  background: linear-gradient(135deg, #6c757d, #5a6268);

  cursor: not-allowed;

  transform: none;

  box-shadow: none;

}

@keyframes fadeInUp {

  from {

    opacity: 0;

    transform: translateY(30px);

  }

  to {

    opacity: 1;

    transform: translateY(0);

  }

}

/\* Responsive adjustments \*/

@media (max-width: 768px) {

  .cart {

    padding: 20px 15px;

    margin: 10px;

  }

  .cart h2 {

    font-size: 2rem;

  }

  .item {

    flex-direction: column;

    text-align: center;

  }

  .image {

    margin-right: 0;

    margin-bottom: 15px;

  }

  .quantityControl {

    justify-content: center;

  }

  .totalSection {

    padding: 20px;

  }

  .orderForm {

    padding: 20px;

  }

}

Листинг 1 – Cart.module.css

.catalog {

  padding: 30px 20px;

  max-width: 1400px;

  margin: 0 auto;

  background: linear-gradient(135deg, #ffffff, #f8f9fa);

  min-height: calc(100vh - 200px);

  border-radius: 15px;

  box-shadow: 0 10px 30px rgba(0,0,0,0.08);

  margin-top: 20px;

  margin-bottom: 20px;

  position: relative;

  overflow: hidden;

}

.catalog::before {

  content: '';

  position: absolute;

  top: 0;

  left: 0;

  right: 0;

  height: 5px;

  background: linear-gradient(90deg, #e91e63, #667eea, #00bcd4);

  animation: gradientShift 3s infinite alternate;

}

@keyframes gradientShift {

  0% { background: linear-gradient(90deg, #e91e63, #667eea, #00bcd4); }

  50% { background: linear-gradient(90deg, #00bcd4, #e91e63, #667eea); }

  100% { background: linear-gradient(90deg, #667eea, #00bcd4, #e91e63); }

}

.catalog h2 {

  text-align: center;

  font-size: 2.5rem;

  margin-bottom: 20px;

  color: #2c3e50;

  text-shadow: 2px 2px 4px rgba(0,0,0,0.1);

  background: linear-gradient(135deg, #e91e63, #667eea);

  -webkit-background-clip: text;

  -webkit-text-fill-color: transparent;

  background-clip: text;

  position: relative;

  display: inline-block;

  margin-bottom: 30px;

}

.catalog h2::after {

  content: '';

  position: absolute;

  bottom: -10px;

  left: 50%;

  transform: translateX(-50%);

  width: 80px;

  height: 3px;

  background: linear-gradient(90deg, #e91e63, #667eea);

  border-radius: 2px;

}

.productsGrid {

  display: grid;

  grid-template-columns: repeat(auto-fill, minmax(280px, 1fr));

  gap: 30px;

  margin-top: 20px;

  padding: 10px;

  animation: fadeInUp 0.8s ease-out;

}

@keyframes fadeInUp {

  from {

    opacity: 0;

    transform: translateY(30px);

  }

  to {

    opacity: 1;

    transform: translateY(0);

  }

}

.catalog p {

  text-align: center;

  font-size: 1.2rem;

  color: #666;

  margin-top: 40px;

  padding: 20px;

  background: linear-gradient(135deg, #f8f9fa, #e9ecef);

  border-radius: 10px;

  border-left: 4px solid #e91e63;

  animation: fadeIn 1s ease-out;

}

@keyframes fadeIn {

  from { opacity: 0; }

  to { opacity: 1; }

}

/\* Responsive grid adjustments \*/

@media (max-width: 768px) {

  .productsGrid {

    grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));

    gap: 20px;

  }

  .catalog h2 {

    font-size: 2rem;

  }

}

@media (max-width: 480px) {

  .productsGrid {

    grid-template-columns: 1fr;

    gap: 20px;

  }

  .catalog h2 {

    font-size: 1.8rem;

  }

  .catalog {

    padding: 20px 15px;

  }

}

Листинг 2 – Catalog.module.css

.container {

  padding: 20px;

  max-width: 1200px;

  margin: 0 auto;

}

.header {

  display: flex;

  justify-content: space-between;

  align-items: center;

  margin-bottom: 20px;

}

.clearBtn {

  background-color: #dc3545;

  color: white;

  border: none;

  padding: 8px 12px;

  border-radius: 4px;

  cursor: pointer;

}

.clearBtn:hover {

  background-color: #c82333;

}

.categoryGroup {

  margin-bottom: 30px;

  border: 1px solid #e0e0e0;

  border-radius: 8px;

  padding: 15px;

  background-color: #fafafa;

}

.categoryTitle {

  margin: 0 0 15px 0;

  padding-bottom: 10px;

  border-bottom: 2px solid #ddd;

  color: #333;

}

.compareTable {

  display: grid;

  grid-template-columns: 200px repeat(3, 1fr);

  gap: 1px;

  background-color: #ddd;

  border: 1px solid #ddd;

  border-radius: 4px;

  overflow-x: auto;

  max-width: 100%;

  min-width: fit-content;

}

.compareTable.two-products {

  grid-template-columns: 200px repeat(2, 1fr);

}

.compareTable.three-products {

  grid-template-columns: 200px repeat(3, 1fr);

}

.compareTable.four-or-more-products {

  grid-template-columns: 200px repeat(4, 1fr);

}

.headerRow, .dataRow {

  display: contents;

}

.propertyCell, .productCell {

  background-color: white;

  padding: 12px;

  border-bottom: 1px solid #ddd;

}

.propertyCell {

  background-color: #f8f9fa;

  font-weight: bold;

  display: flex;

  align-items: center;

}

.productCell {

  text-align: center;

}

.productImage {

  width: 80px;

  height: 80px;

  object-fit: cover;

  margin-bottom: 8px;

  display: block;

  margin-left: auto;

  margin-right: auto;

}

.removeProductBtn {

  background-color: #dc3545;

  color: white;

  border: none;

  padding: 4px 8px;

  border-radius: 4px;

  cursor: pointer;

  font-size: 12px;

  margin-top: 8px;

  width: 100%;

}

.removeProductBtn:hover {

  background-color: #c82333;

}

/\* Responsive adjustments \*/

@media (max-width: 768px) {

  .container {

    padding: 10px;

  }

  .header {

    flex-direction: column;

    align-items: flex-start;

    gap: 10px;

  }

  .compareTable {

    display: block;

    overflow-x: auto;

    grid-template-columns: none;

    max-width: 100%;

    min-width: fit-content;

  }

  .headerRow, .dataRow {

    display: flex;

    min-width: max-content;

  }

  .propertyCell, .productCell {

    min-width: 150px;

    flex: 0 0 auto;

    padding: 8px;

  }

  .productImage {

    width: 60px;

    height: 60px;

  }

  .removeProductBtn {

    font-size: 10px;

    padding: 2px 4px;

  }

}

/\* Additional responsive adjustments for smaller screens \*/

@media (max-width: 480px) {

  .propertyCell, .productCell {

    min-width: 120px;

    padding: 6px;

    font-size: 12px;

  }

  .productImage {

    width: 50px;

    height: 50px;

  }

  .productCell h4 {

    font-size: 12px;

    margin: 5px 0;

  }

  .removeProductBtn {

    font-size: 9px;

    padding: 2px 3px;

  }

}

Листинг 3 – Compare.module.css

.container {

  padding: 20px;

  max-width: 1200px;

  margin: 0 auto;

}

.favoritesGrid {

  display: grid;

  grid-template-columns: repeat(auto-fill, minmax(280px, 1fr));

  gap: 25px;

  margin-top: 20px;

}

.favoriteItem {

  border: 1px solid #eee;

  border-radius: 12px;

  padding: 16px;

  display: flex;

  flex-direction: column;

  align-items: center;

  transition: all 0.3s ease;

  background: linear-gradient(135deg, #ffffff, #f8f9fa);

  box-shadow: 0 4px 6px rgba(0,0,0,0.05);

  position: relative;

  overflow: hidden;

  min-width: 250px;

}

.favoriteItem:hover {

  box-shadow: 0 8px 25px rgba(0,0,0,0.15);

  transform: translateY(-5px);

}

.favoriteItem::before {

  content: '';

  position: absolute;

  top: 0;

  left: 0;

  right: 0;

  height: 3px;

  background: linear-gradient(90deg, #e91e63, #667eea);

  transition: all 0.3s ease;

}

.favoriteItem:hover::before {

  height: 5px;

}

.image {

  width: 160px;

  height: 160px;

  object-fit: cover;

  margin-bottom: 12px;

  border-radius: 8px;

  transition: transform 0.3s ease;

}

.favoriteItem:hover .image {

  transform: scale(1.05);

}

.info {

  text-align: center;

  width: 100%;

}

.title {

  font-size: 18px;

  font-weight: bold;

  margin-bottom: 8px;

  color: #2c3e50;

  transition: color 0.3s ease;

  margin-top: 0;

}

.favoriteItem:hover .title {

  color: #e91e63;

}

.price {

  color: #e91e63;

  font-weight: 600;

  margin: 8px 0;

  font-size: 18px;

}

.category {

  color: #007bff;

  font-size: 14px;

  margin-bottom: 8px;

  font-weight: 500;

}

.character {

  color: #666;

  font-size: 14px;

  margin-bottom: 8px;

}

.removeBtn {

  background: linear-gradient(135deg, #dc3545, #bd2130);

  color: white;

  border: none;

  padding: 12px 16px;

  border-radius: 25px;

  cursor: pointer;

  margin-top: 10px;

  width: 100%;

  font-weight: 500;

  transition: all 0.3s ease;

  box-shadow: 0 4px 8px rgba(220, 53, 69, 0.3);

  max-width: 200px;

  margin: 8px auto;

}

.removeBtn:hover {

  background: linear-gradient(135deg, #bd2130, #9d1e2a);

  transform: translateY(-2px);

  box-shadow: 0 6px 12px rgba(220, 53, 69, 0.4);

}

.addToCartBtn {

  background: linear-gradient(135deg, #28a745, #218838);

  color: white;

  border: none;

  padding: 12px 16px;

  border-radius: 25px;

  cursor: pointer;

  margin-top: 10px;

  width: 100%;

  max-width: 200px;

  font-weight: 500;

  transition: all 0.3s ease;

  box-shadow: 0 4px 8px rgba(40, 167, 69, 0.3);

  margin: 8px auto 0;

}

.addToCartBtn:hover {

  background: linear-gradient(135deg, #218838, #1e7e34);

  transform: translateY(-2px);

  box-shadow: 0 6px 12px rgba(40, 167, 69, 0.4);

}

.buttonGroup {

  display: flex;

  flex-direction: column;

  gap: 8px;

  width: 100%;

  align-items: center;

  margin-top: 10px;

}

/\* Responsive grid adjustments \*/

@media (max-width: 768px) {

  .favoritesGrid {

    grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));

    gap: 20px;

  }

  .favoriteItem {

    min-width: 200px;

    padding: 12px;

  }

  .image {

    width: 140px;

    height: 140px;

  }

  .title {

    font-size: 16px;

  }

  .price {

    font-size: 16px;

  }

  .removeBtn, .addToCartBtn {

    padding: 10px 14px;

    font-size: 14px;

    max-width: 100%;

  }

}

@media (max-width: 480px) {

  .favoritesGrid {

    grid-template-columns: 1fr;

    gap: 20px;

  }

  .favoriteItem {

    min-width: 160px;

    padding: 10px;

  }

  .image {

    width: 120px;

    height: 120px;

  }

  .title {

    font-size: 15px;

  }

  .price, .category {

    font-size: 13px;

  }

  .removeBtn, .addToCartBtn {

    padding: 8px 12px;

    font-size: 13px;

    max-width: 100%;

  }

}

Листинг 4 – favourites.module.css

.filters {

  background: #f8f9fa;

  padding: 20px;

  border-radius: 8px;

  box-shadow: 0 2px 4px rgba(0,0,0,0.1);

  margin-bottom: 20px;

}

.filterGroup {

  margin-bottom: 15px;

}

.filterGroup label {

  display: block;

  margin-bottom: 5px;

  font-weight: bold;

  color: #333;

}

.filterGroup select,

.filterGroup input {

  width: 100%;

  padding: 8px;

  border: 1px solid #ddd;

  border-radius: 4px;

  font-size: 14px;

}

.searchContainer {

  display: flex;

  gap: 5px;

}

.searchContainer input {

  flex: 1;

}

.searchContainer button,

.priceInputs button {

  padding: 8px 12px;

  background: #007bff;

  color: white;

  border: none;

  border-radius: 4px;

  cursor: pointer;

}

.searchContainer button:hover,

.priceInputs button:hover {

  background: #0056b3;

}

.priceInputs {

  display: flex;

  align-items: center;

  gap: 5px;

}

.priceInputs input {

  flex: 1;

}

.priceInputs span {

  white-space: nowrap;

}

.resetButton {

  width: 100%;

  padding: 10px;

  background: #6c757d;

  color: white;

  border: none;

  border-radius: 4px;

  cursor: pointer;

}

.resetButton:hover {

  background: #545b62;

}

/\* Responsive adjustments \*/

@media (max-width: 768px) {

  .filters {

    padding: 15px;

  }

  .priceInputs {

    flex-direction: column;

    align-items: stretch;

  }

  .priceInputs span {

    align-self: flex-start;

    margin-bottom: 5px;

  }

  .searchContainer {

    flex-direction: column;

  }

  .searchContainer button {

    width: 100%;

  }

}

@media (max-width: 480px) {

  .filters {

    padding: 10px;

  }

  .filterGroup label {

    font-size: 14px;

  }

  .filterGroup select,

  .filterGroup input {

    font-size: 13px;

  }

}

Листинг 5 – Filters.module.css

:root{

  --accent:#ff6b6b;

  --accent-2:#6b8cff;

  --bg:#0f1724;

  --card:#0b1220;

  --muted:#98a0b3;

  --glass: rgba(255,255,255,0.04);

}

\* { box-sizing: border-box; }

body {

  margin:0;

  font-family: Inter, 'Segoe UI', Roboto, sans-serif;

  background: linear-gradient(180deg,#071028 0%, #071a2a 100%);

  color:#e6eef8;

  -webkit-font-smoothing:antialiased;

  -moz-osx-font-smoothing:grayscale;

}

/\* card \*/

.card {

  background: linear-gradient(180deg, rgba(255,255,255,0.03), rgba(0,0,0,0.04));

  border-radius: 14px;

  padding: 16px;

  box-shadow: 0 6px 30px rgba(8,12,20,0.6);

  border: 1px solid rgba(255,255,255,0.03);

}

/\* buttons \*/

.btn {

  display:inline-flex; align-items:center; gap:8px;

  padding:10px 14px; border-radius:10px; border:none; cursor:pointer;

  background: linear-gradient(90deg,var(--accent),var(--accent-2));

  color:white; font-weight:600;

  transition: transform .15s ease, box-shadow .15s;

}

.btn:hover{ transform: translateY(-3px); box-shadow: 0 12px 30px rgba(107,107,255,0.12); }

/\* плавный вход секций \*/

.fadeUp {

  opacity:0; transform: translateY(12px) scale(.995);

  animation: fadeUp .5s forwards;

}

@keyframes fadeUp {

  to { opacity:1; transform: translateY(0) scale(1); }

}

/\* карточки при наведении \*/

.cardHover {

  transition: transform .2s cubic-bezier(.2,.8,.2,1), box-shadow .2s;

}

.cardHover:hover { transform: translateY(-8px) rotate(-0.5deg); box-shadow: 0 18px 50px rgba(0,0,0,0.6); }

Листинг 6 – globals.module.css

.header {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 12px 24px;

  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

  border-bottom: 1px solid #eee;

  box-shadow: 0 2px 10px rgba(0,0,0,0.1);

  position: sticky;

  top: 0;

  z-index: 100;

}

.logoBlock {

  display: flex;

  align-items: center;

  text-decoration: none;

  color: inherit;

}

.logo {

  height: 48px;

  cursor: pointer;

  animation: pulse 2s infinite;

}

@keyframes pulse {

  0% { transform: scale(1); }

  50% { transform: scale(1.05); }

  100% { transform: scale(1); }

}

.searchBlock {

  display: flex;

  flex: 1;

  max-width: 500px;

  margin: 0 24px;

  position: relative;

}

.searchInput {

  flex: 1;

  padding: 12px 16px;

  border: 2px solid #e0e0e0;

  border-radius: 25px;

  font-size: 16px;

  outline: none;

  transition: all 0.3s ease;

  background-color: rgba(255, 255, 255, 0.9);

}

.searchInput:focus {

  border-color: #e91e63;

  box-shadow: 0 0 0 3px rgba(233, 30, 99, 0.2);

  background-color: white;

}

.searchButton {

  padding: 12px 20px;

  background: linear-gradient(135deg, #e91e63, #c2185b);

  color: white;

  border: none;

  border-radius: 0 25px 25px 0;

  cursor: pointer;

  font-size: 16px;

  transition: all 0.3s ease;

}

.searchButton:hover {

  background: linear-gradient(135deg, #c2185b, #ad1457);

  transform: scale(1.05);

}

.actions {

  display: flex;

  gap: 16px;

  position: relative;

}

.iconButton {

  background: rgba(255, 255, 255, 0.2);

  border: none;

  cursor: pointer;

  border-radius: 50%;

  width: 50px;

  height: 50px;

  display: flex;

  align-items: center;

  justify-content: center;

  position: relative;

  transition: all 0.3s ease;

  backdrop-filter: blur(10px);

  border: 1px solid rgba(255, 255, 255, 0.3);

}

.iconButton:hover {

  background: rgba(255, 255, 255, 0.3);

  transform: scale(1.1);

}

.iconButton img {

  height: 24px;

  transition: all 0.3s ease;

}

.iconButton:hover img {

  transform: scale(1.2);

}

.heartFilled {

  filter: drop-shadow(0 0 5px rgba(233, 30, 99, 0.5));

  animation: heartBeat 1.5s infinite;

}

@keyframes heartBeat {

  0% { transform: scale(1); }

  25% { transform: scale(1.1); }

  50% { transform: scale(1); }

  75% { transform: scale(1.05); }

  100% { transform: scale(1); }

}

.badge {

  position: absolute;

  top: -8px;

  right: -8px;

  background: linear-gradient(135deg, #ff4757, #ff3838);

  color: white;

  border-radius: 50%;

  width: 22px;

  height: 22px;

  display: flex;

  align-items: center;

  justify-content: center;

  font-size: 12px;

  font-weight: bold;

  box-shadow: 0 2px 6px rgba(0,0,0,0.2);

  animation: badgePulse 2s infinite;

}

@keyframes badgePulse {

  0% { transform: scale(1); }

  50% { transform: scale(1.1); }

  100% { transform: scale(1); }

}

.authMark {

  position: absolute;

  bottom: 0;

  right: 0;

  width: 12px;

  height: 12px;

  background-color: #2ecc71;

  border-radius: 50%;

  border: 2px solid white;

}

/\* Responsive adjustments for mobile \*/

@media (max-width: 768px) {

  .header {

    padding: 10px 15px;

    flex-wrap: wrap;

    gap: 10px;

  }

  .logo {

    height: 40px;

  }

  .searchBlock {

    order: 3; /\* Move search to the bottom on mobile \*/

    max-width: 100%;

    margin: 10px 0 0;

  }

  .searchInput {

    padding: 10px 15px;

    font-size: 14px;

  }

  .searchButton {

    padding: 10px 15px;

    font-size: 14px;

  }

  .actions {

    gap: 10px;

  }

  .iconButton {

    width: 44px;

    height: 44px;

  }

  .iconButton img {

    height: 20px;

  }

}

@media (max-width: 480px) {

  .header {

    padding: 8px 10px;

  }

  .logo {

    height: 35px;

  }

  .iconButton {

    width: 40px;

    height: 40px;

  }

  .badge {

    width: 18px;

    height: 18px;

    font-size: 10px;

  }

}

/\* Search results styles \*/

.searchResults {

  position: absolute;

  top: 100%;

  left: 0;

  right: 0;

  background: white;

  border: 1px solid #e0e0e0;

  border-radius: 8px;

  box-shadow: 0 4px 12px rgba(0,0,0,0.15);

  z-index: 99;

  max-height: calc(33vh - 60px); /\* Maximum height is one-third of viewport height minus some offset \*/

  overflow-y: auto; /\* Enable vertical scrolling when content overflows \*/

  margin-top: 5px;

}

.searchResultItem {

  padding: 12px 16px;

  cursor: pointer;

  border-bottom: 1px solid #f0f0f0;

  transition: background-color 0.2s ease;

}

.searchResultItem:last-child {

  border-bottom: none;

}

.searchResultItem:hover {

  background-color: #f9f9f9;

}

.productResult {

  display: flex;

  align-items: center;

  gap: 12px;

}

.resultImage {

  width: 40px;

  height: 40px;

  object-fit: cover;

  border-radius: 4px;

}

.resultInfo {

  flex: 1;

}

.resultTitle {

  font-weight: 500;

  color: #333;

  margin-bottom: 4px;

  white-space: nowrap;

  overflow: hidden;

  text-overflow: ellipsis;

}

.resultPrice {

  font-weight: bold;

  color: #e91e63;

  font-size: 14px;

}

.categoryResult {

  display: flex;

  align-items: center;

  gap: 8px;

  font-weight: 500;

  color: #555;

}

Листинг 7 – header.module.css