МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОМПЬЮТЕРНЫХ НАУК
КАФЕДРА АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ И ДИЗАЙНА
НАПРАВЛЕНИЕ 09.04.02 ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

на тему: Исследование и разработка пользовательского интерфейса E-commerce платформы на Next.js

Студент _____

Руководитель работы _____ *Е.Г. Коржов*

Нормоконтроль проведен _____ *А.С. Оганесян*

Проверка на заимствования проведена _____ *А.А. Петрыкина*

Работа рассмотрена кафедрой и допущена к защите в ГЭК

Заведующий кафедрой _____ *Е.Г. Коржов*

Директор института _____ *С.В. Солодов*

Москва 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

УТВЕРЖДАЮ

Институт Информационных технологий и компьютерных наук
Кафедра Автоматизированного проектирования и дизайна _____ **Зав. кафедрой** _____ Е.Г. Коржов
Направление 09.04.02 Информационные системы и технологии _____ «_____»_____2025г.

З А Д А Н И Е
Н А В Ы П О Л Н Е Н И Е В Ы П У С К Н О Й К В А Л И Ф И К А Ц И О Н Н О Й Р А Б О Т Ы М А Г И С Т Р А

Студенту группы _____
(Ф.И.О. полностью)

1. Тема работы Исследование и разработка пользовательского интерфейса E-commerce платформы на Next.js
2. Цели работы Разработка и оптимизация пользовательского интерфейса платформы электронной коммерции с использованием современных технологий веб-разработки, обеспечивающих высокую производительность, удобство использования и соответствие макету.

3. Исходные данные Макет пользовательского интерфейса в Figma _____

4. Основная литература, в том числе: Монографии, учебники и т.п. 1. Next.js documentation. Vercel, 2023 [Электронный ресурс]. URL: https://nextjs.org/docs 2. The Next.js Handbook: A Complete Resource for Developers. – Ares Lnc, 2023. 3. Дон Норман. “The Design of Everyday Things”. – М.: “Манн, Иванов и Фербер”, 2018 – 384 с._____

4.1. Отчеты по НИР, диссертации, дипломные проекты и т.п. Беляев Н.Ю., “Разработка веб-платформы управления и публикации научных статей”, ВКР, 2023 _____

4.2. Периодическая литература Рекомендации по мобильной индексации [Электронный ресурс] Google Developers. URL: https://developers.google.com/search/docs/crawling-indexing/mobile/mobile-sites-mobile-first-indexing.

4.3. Патенты _____

4.4. Справочники и методическая литература (в том числе литература по методам обработки экспериментальных данных) _____

_____

5. Перечень основных этапов исследования и форма промежуточной отчетности по каждому этапу Исследование и развитие E-commerce платформы (отчет по НИР). Исследование специфики развития E-commerce платформы (отчет по преддипломной практике).

6. Аппаратура и методики, которые должны быть использованы при проведении исследований Персональный компьютер с установленным Node.js, редактором кода (VS Code), браузером Chrome для тестирования.

7. Использование информационных технологий при проведении исследований Фреймворк Next.js, язык программирования TypeScript, фреймворк Tailwind CSS, система контроля версий Git, редактор кода VS Code, сервис Figma для создания прототипов и макетов, инструменты разработчика Chrome DevTools

_____

8. Перечень подлежащих разработке вопросов по экономике НИР _____

_____

_____

Согласовано: не предусмотрено **Консультант по экономике**

9. Перечень подлежащих разработке вопросов по безопасности жизнедеятельности _____

_____

_____

Согласовано: не предусмотрено **Консультант по безопасности жизнедеятельности**

10. Перечень подлежащих разработке вопросов по охране окружающей среды _____

_____

Согласовано: не предусмотрено **Консультант по охране окружающей среды**

11. Перечень (примерный) основных вопросов, которые должны быть рассмотрены и проанализированы в литературном обзоре _____

12. Перечень (примерный) иллюстрированного материала скриншоты, иллюстрации

13. Руководитель диссертации _____ доцент, к.т.н. Коржов Евгений Геннадьевич _____
(Должность, звание, ф.и.о.)

(подпись)
_____

14. Консультанты (с указанием относящихся к ним разделов) _____

_____

Дата выдачи задания 03.03.2025

Задание принял к исполнению студент _____
(подпись)

Аннотация

Выпускная квалификационная работа (ВКР) содержит: 83 страницы, 3 раздела, 10 рисунков, 2 таблицы, 30 литературных источников, 1 приложение. В работе рассмотрены современные технологии фронтенд-разработки, реализован интерфейс веб-приложения интернет-магазина на Next.js с использованием TypeScript и Tailwind CSS, проведена интеграция с серверной частью через REST API.

Выпускная квалификационная работа состоит из пояснительной записки, графического приложения и презентации.

Пояснительная записка содержит введение, три раздела и заключение.

Первый раздел посвящен теоретическому обзору объекта исследования.

Во втором разделе подробно описывается поэтапный процесс разработки и реализации интерфейса страницы редактирования профиля пользователя и страницы товара, включающий адаптивную верстку компонентов и обеспечение корректного взаимодействия фронтенда с сервером.

В заключении подводится итог проведенного исследования.

Annotation

This graduation thesis is dedicated to the development and implementation of a user interface for a web application using modern frontend technologies. The goal of the project is to create a functional, responsive, and easily extendable interface that ensures a convenient user experience for an online store system. To achieve this goal, the Next.js framework, the TypeScript programming language, and the Tailwind CSS utility-first framework were used, along with the implementation of integration mechanisms with the backend via a REST API.

As part of the project, the interface for the user profile editing page was designed and implemented, adaptive layout of key components was carried out, and correct interaction between the frontend and backend was ensured. Interface integration with the server side was completed, testing was performed, and an analytical report was prepared based on the results. The thesis is presented on 83 pages and includes 10 figures, 2 tables, and a list of 30 references.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность работы. Современная цифровая экономика активно развивается в направлении онлайн-торговли. Электронная коммерция стала неотъемлемой частью повседневной жизни миллионов пользователей. На фоне роста конкуренции на рынке интернет-магазинов особое значение приобретает качество пользовательского интерфейса (UI, от англ. user interface - интерфейс пользователя), от которого напрямую зависит удобство, лояльность и конверсия посетителей в покупателей. Оптимизация пользовательского интерфейса позволяет повысить скорость взаимодействия с платформой, улучшить восприятие информации, упростить навигацию и, как следствие, увеличить коммерческую эффективность. Таким образом, тема работы является актуальной в контексте развития высококачественных, ориентированных на пользователя интерфейсов в E-commerce (англ. E-commerce или electronic commerce - электронная торговля).

Объектом исследования данной работы выступает пользовательский интерфейс веб-приложения электронной коммерции, который включает в себя функционал для аутентификации пользователей, управления списками товаров, оформления покупок через корзину и проведения платежей посредством интегрированной системы. Платформа представляет собой набор программных решений и технологий, обеспечивающих возможность осуществления торговых операций в онлайн-формате. В рамках исследования основное внимание сосредоточено на анализе и совершенствовании ключевых компонентов таких платформ.

Предмет исследования - это методы и технологии оптимизации интерфейса E-commerce платформы с использованием современных инструментов веб-разработки.

Цель исследования - разработка и оптимизация пользовательского интерфейса интернет-магазина с использованием современных технологий

веб-разработки, обеспечивающих высокую производительность, удобство использования и соответствие макету.

Задачи исследования:

1. Проанализировать современные подходы к созданию пользовательских интерфейсов для E-commerce.
2. Исследовать возможности современных инструментов веб-разработки для проектирования интерфейса.
3. Реализовать интерфейс по предоставленному макету с учетом адаптивности.
4. Провести пользовательское тестирование и оценку производительности.
5. Выполнить итерационную оптимизацию на основе собранной обратной связи.

Практическая значимость работы заключается в разработке и тестировании полноценного интерфейса E-commerce платформы, пригодного для решения бизнес-задач. Необходимость изучения обусловлена динамичным ростом объемов онлайн-продаж как в мировом масштабе, так и в России. По данным аналитиков, ежегодное увеличение оборота в данном секторе связано с изменениями потребительских предпочтений, направленных на использование цифровых каналов для покупок. Дополнительно пандемия COVID-19 ускорила процесс адаптации компаний к онлайн-торговле, усилив конкуренцию в этом сегменте. Например, в России объемы продаж в сфере электронной коммерции растут в среднем на 20% в год [1], и, согласно прогнозам, в 2025 году они достигнут 14900 миллиардов рублей [2]. Таким образом, анализ и развитие платформ E-commerce являются актуальными задачами для развития электронной торговли.

Раздел 1. Анализ современного состояния разработки пользовательских интерфейсов для E-commerce платформ

1.1 Задачи и особенности пользовательского интерфейса в E-commerce

Электронная коммерция (E-commerce) представляет собой процесс купли-продажи товаров и услуг с использованием сети интернет [3]. Это направление активно развивается и играет важную роль в современной экономике, связывая продавцов и покупателей через удобные цифровые платформы. Современные платформы электронной коммерции представляют собой сложные программные комплексы, в которых все компоненты тесно взаимосвязаны и обеспечивают беспрерывную и эффективную работу онлайн-магазина. Центральным элементом таких систем является пользовательский интерфейс.

Основная задача пользовательского интерфейса заключается в обеспечении успешного взаимодействия между пользователем и платформой. В условиях высокой конкуренции на рынке электронной торговли именно удобство, понятность и отзывчивость интерфейса становятся решающими факторами для удержания пользователей и повышения конверсии. Согласно исследованию Forrester Research, хорошо продуманный пользовательский интерфейс может повысить коэффициент конверсии до 200%, а улучшенный пользовательский опыт (UX, от англ. user expirience - пользовательский опыт) - до 400% [4].

Уже на этапе знакомства с главной страницей платформы электронной коммерции пользователь оценивает внешний вид, структуру и простоту навигации, что в значительной степени влияет на его решение остаться или покинуть ресурс. Эстетика, цветовая палитра, типографика и анимация должны быть не просто современными, но и соответствовать ожиданиям

целевой аудитории, формируя доверие и эмоциональную привязанность к бренду [5].

Навигация по платформе должна быть максимально логичной и простой. Пользователь не должен тратить усилия на то, чтобы разобраться в устройстве меню или найти нужную категорию товаров. Четкая маршрутизация, единообразие элементов управления и предсказуемость действий интерфейса способствуют снижению когнитивной нагрузки и ускоряют принятие решений. Особенно важно обеспечить такой опыт на мобильных устройствах, доля которых в глобальных интернет-продажах превышает 70%, согласно данным Statista.

Ключевая цель любого интерфейса в E-commerce - быстро и удобно довести пользователя до покупки. Это означает минимизацию количества шагов в пользовательском пути: от выбора товара до оформления заказа и оплаты. Интерфейс должен обеспечивать все необходимые функции, не создавая при этом ощущения перегруженности или сложности.

Современный UI в электронной коммерции также должен быть адаптивным - одинаково хорошо работать на всех типах устройств и разрешениях экранов. Наряду с этим возрастает значение доступности интерфейсов, обеспечивающей возможность полноценного взаимодействия с сайтом для людей с ограниченными возможностями. Руководства по доступности, такие как WCAG (Web Content Accessibility Guidelines), становятся важными стандартами при проектировании интерфейсов [6].

Обратная связь играет не менее важную роль - каждый клик, действие или изменение должны сопровождаться визуальными или текстовыми откликами со стороны системы. Это позволяет пользователю чувствовать контроль над происходящим и повышает его уверенность в платформе.

Наконец, важнейшей задачей является формирование доверия. Особенно критично это для новых клиентов, которые впервые взаимодействуют с брендом онлайн. Наличие отзывов, рейтингов, подробной информации о товаре и условиях доставки, а также прозрачность процесса

возврата и надежные способы оплаты значительно увеличивают лояльность и готовность пользователя к совершению покупок. Все это делает интерфейс не просто удобным инструментом взаимодействия, а важнейшим фактором успешности E-commerce проекта.

К проектированию пользовательского интерфейса для электронной коммерции предъявляют особые требования, продиктованные необходимостью обеспечивать удобство работы с большим объемом данных. Каталоги товаров включают изображения, цены, акции, рейтинги и другие сведения, которые важно структурировать, чтобы избежать визуальной перегрузки. Интерфейс должен быть интерактивным - фильтры, корзина, сортировка товаров и рекомендательные блоки требуют динамичного поведения, зачастую без полной перезагрузки страницы, что достигается за счет клиентских фреймворков и асинхронных запросов к серверу.

Кроме того, пользовательский интерфейс должен тесно связан с бизнес-логикой платформы. В это понятие входит корректное отображение скидок, статусов заказов, взаимодействие с личным кабинетом пользователя и гибкость к изменениям бизнес-правил. Важно также учитывать требования поисковой SEO (SEO, от англ. search engine optimization – поисковая оптимизация) оптимизации и производительности. Интерфейс должен быть оптимизирован для быстрой загрузки страниц, особенно на мобильных устройствах. Использование серверного рендеринга, например, через фреймворк Next.js, позволяет добиться более высокой видимости в поисковых системах и улучшает восприятие со стороны пользователей (Next.js SEO Benefits).

Также интерфейс должен без проблем интегрироваться с внешними сервисами - платежными системами, аналитикой, чатами поддержки и другими инструментами. Все эти элементы необходимо гармонично встраивать в общую архитектуру интерфейса, сохраняя его целостность и удобство использования.

Таким образом, эффективный пользовательский интерфейс в электронной коммерции - это результат продуманного сочетания визуального дизайна, технической реализации, пользовательских сценариев и бизнес-целей. Это неотъемлемый компонент, определяющий успешность онлайн-торговли и степень удовлетворенности клиента на всех этапах его взаимодействия с платформой.

Психологические аспекты восприятия интерфейса оказывают прямое влияние на поведение пользователя. Согласно исследованиям, большинство пользователей принимают решение остаться или покинуть сайт в течение первых 10–15 секунд. Это означает, что интерфейс должен быстро загрузиться и моментально вызвать ощущение удобства и профессионализма.

Кроме того, исследования UX показывают, что даже небольшие улучшения в пользовательском опыте (например, улучшение адаптивности, сокращение времени загрузки страниц, упрощение оформления заказа) могут привести к значительному росту коэффициента конверсии. Также важно учитывать пользовательские сценарии: поведение постоянного клиента отличается от поведения нового пользователя. Интерфейс должен быть одинаково удобен и для одного, и для другого, поддерживая персонализацию.

Таким образом, пользовательский интерфейс в E-commerce - это не просто “визуальная оболочка”, а сложная система, влияющая на коммерческий успех проекта.

1.2 Обзор современных технологий фронтенд-разработки

Развитие веб-технологий за последнее десятилетие привело к появлению обширного набора инструментов и подходов к созданию пользовательских интерфейсов. Особенно стремительное развитие наблюдается в области веб-разработки, где производительность, масштабируемость и качество взаимодействия с пользователем напрямую

влияют на успех цифрового продукта. В контексте электронной коммерции эти аспекты приобретают решающее значение, поскольку каждый элемент пользовательского интерфейса влияет на удобство покупателя, а, следовательно, и на коммерческие показатели платформы.

Веб-разработка в современной парадигме представляет собой проектирование, реализацию и оптимизацию интерфейсов с использованием языков программирования, фреймворков, систем сборки, компонентных библиотек и макетных редакторов. Рассмотрим ключевые технологии, наиболее широко применяемые сегодня в профессиональной веб-разработке.

1.2.1 Языки и фреймворки веб-разработки

Традиционно считается, что основными инструментами веб-разработки являются HTML, CSS и JavaScript.

HTML (от англ. HyperText Markup Language - язык гипертекстовой разметки) является основным языком разметки, определяющим структуру веб-страницы. Он используется для описания таких элементов, как заголовки, абзацы, списки, изображения, формы и таблицы. Современные стандарты HTML5 расширили возможности языка, добавив встроенные API (от англ, Application Programming Interface – интерфейс программирования приложения) и улучшенную поддержку мультимедиа и семантики.

CSS (от англ. Cascading Style Sheets - каскадные таблицы стилей) отвечает за визуальное оформление HTML-элементов. В современных проектах используются методологии, такие как BEM (от англ. Block Element Modifier – Блок, Элемент, Модификатор), и инструменты, такие как препроцессоры (SCSS, LESS), а также утилитарные фреймворки вроде Tailwind CSS, которые упрощают управление стилями и повышают гибкость верстки.

JavaScript является основным языком программирования на стороне клиента, обеспечивающим интерактивность и динамичность интерфейса.

Благодаря JavaScript стало возможным создавать одностраничные приложения SPA (от англ. Single Page Application – одностраничное веб-приложение), обновлять контент без перезагрузки страницы, управлять состоянием приложения, взаимодействовать с API и обрабатывать пользовательские события.

Одним из самых значимых улучшений JavaScript является язык TypeScript - надмножество JavaScript, добавляющее строгую типизацию. Использование TypeScript значительно повышает надежность и предсказуемость кода благодаря системе типов, поддержке автодополнения и статическому анализу кода на этапе разработки.

TypeScript способствует повышению качества разработки за счет:

- раннего выявления ошибок;
- улучшенной читаемости и документированности кода;
- упрощения командной разработки;
- ускорения процесса тестирования и отладки.

В проектах на базе фреймворка Next.js использование языка TypeScript становится стандартом.

Современная фронтенд-разработка опирается на использование фреймворков и библиотек компонентов, предоставляющих мощные инструменты для построения UI.

React - одна из самых популярных JavaScript-библиотек [7]. Она предлагает компонентный подход к построению пользовательского интерфейса, где каждый элемент представляется в виде отдельного логически завершенного компонента. React активно применяется в E-commerce благодаря своей гибкости, виртуальному DOM дереву и высокой производительности.

1.2.2 Преимущества Next.js для создания интерфейсов

Next.js представляет собой фреймворк, построенный на основе React, и предлагает расширенные возможности для разработки пользовательских интерфейсов. Одним из ключевых преимуществ Next.js является поддержка серверного рендеринга SSR (от англ. Server Side Rendering - рендеринг на стороне сервера) и статической генерации SSG (от англ. Static Site Generation - статическая генерация страниц), что положительно сказывается как на скорости загрузки страниц, так и на их способности индексироваться поисковыми системами. Это особенно важно для E-commerce платформ, где SEO играет значимую роль в привлечении трафика.

Фреймворк Next.js также предоставляет встроенную систему маршрутизации, что упрощает создание структуры сайта и навигацию между страницами без необходимости в дополнительной доработке. Это делает разработку более быстрой и прогнозируемой. Фреймворк имеет полную совместимость с TypeScript, позволяя использовать строгую типизацию прямо “из коробки”, что повышает надежность кода и облегчает масштабирование проекта.

Дополнительным преимуществом является возможность реализации серверной логики с помощью встроенных API. Это позволяет объединить фронтенд и бэкенд в едином проекте, обеспечивая гибкость при разработке пользовательских сценариев и взаимодействий с базой данных или внешними сервисами. Кроме того, фреймворк Next.js включает множество инструментов для автоматической оптимизации производительности: разделение кода, предварительная загрузка страниц, адаптивная загрузка изображений и шрифтов - все это направлено на повышение скорости работы интерфейса и улучшение пользовательского опыта.

Благодаря таким возможностям фреймворк Next.js стал одним из наиболее популярных решений для создания современных,

высокопроизводительных и SEO-оптимизированных интерфейсов в сфере электронной коммерции.

1.2.3 Сравнительный анализ фреймворков

В последние годы веб-разработка развивалась в пользу фреймворков, которые позволяют разработчикам создавать быстрые и масштабируемые приложения. Среди них фреймворк Next.js приобрел значительную популярность благодаря своим возможностям гибридного рендеринга, файловой системе маршрутизации и интеграции с React [8]. Тем не менее существует несколько альтернативных фреймворков, каждый из которых имеет свои сильные стороны и компромиссы. В таблице 1 представлен сравнительный анализ нескольких основных фреймворков - Remix, Gatsby, Astro, SvelteKit и Nuxt.js, с оценкой их по ключевым критериям.

Таблица 1 – Сравнительный анализ фреймворков

Фреймворк	Библиотека	SSR	SSG	Производительность	Масштабируемость	Особенности
Next.js	React	+	+	Высокая	Высокая	ISR, API Routes, Middleware, APP Router
Remix	React	+	-	Очень высокая	Высокая	Продвинутый data loading
Nuxt.js	Vue	+	+	Высокая	Высокая	Аналог Next.js на Vue
SvelteKit	Svelte	+	+	Очень высокая	Средняя	Легковесный
Gatsby	React	_	+	Хорошая	Средняя	Интеграция с GraphQL
Astro	React/Vue/ Svelte	+	+	Очень высокая	Средняя	Island architecture
Bkitz.js	React	+	+	Средняя	Средняя	Fullstack

Next.js предлагает гибридную модель рендеринга, которая состоит из статической генерации страниц (SSG), рендеринга на стороне сервера (SSR), инкрементальной статической регенерации (ISR, от англ. Incremental Static Regeneration – инкрементальная статическая регенерация) и рендеринга на стороне клиента (CSR, от англ. Client-Side Rendering – рендеринг на стороне клиента). Эта гибкость позволяет разработчикам оптимизировать отдельные страницы для производительности и SEO в зависимости от конкретных потребностей.

Фреймворк Remix делает акцент на SSR с помощью расширенной стратегии загрузки данных, которая позволяет выполнять параллельную выборку данных и более детальный контроль над рендерингом [9]. Такая модель повышает производительность и улучшает пользовательский опыт за счет сокращения времени до первого байта TTFB (от англ. Time To First Byte - время до первого байта).

Фреймворк Gatsby также делает акцент на статической генерации страниц SSG, в значительной степени полагаясь на GraphQL для создания насыщенных данными предварительно отрендеренных сайтов. Хотя это обеспечивает отличную производительность, время сборки может стать непомерно долгим для более крупных проектов [10].

Фреймворк Astro использует подход “островной архитектуры”, при котором только те части страницы, с которыми взаимодействует пользователь загружаются как интерактивные, тем самым уменьшая полезную нагрузку JavaScript и повышая скорость загрузки страниц [11].

Фреймворки SvelteKit и Nuxt.js также поддерживают гибридный рендеринг. SvelteKit извлекает выгоду из преимуществ производительности компилятора Svelte, который устраняет большую часть накладных расходов во время выполнения, обнаруженных в фреймворках на основе виртуального DOM, таких как React [12]. Фреймворк Nuxt.js, с другой стороны, предлагает аналогичный Next.js процесс разработки, но адаптированный для экосистемы

Vue.js, что делает его сильным кандидатом для разработчиков, ориентированных на Vue.js [13].

Каждый фреймворк лучше подходит для определенных сценариев. Next.js и Remix хорошо подходят для полнофункциональных приложений, которые требуют аутентификации, динамической маршрутизации и серверной логики. Gatsby и Astro лучше подходят для сайтов с большим объемом контента, таких как блоги и маркетинговые страницы, где производительность во время сборки и SEO имеют первостепенное значение. SvelteKit привлекателен для разработчиков, ищущих высокую производительность и простоту, в то время как Nuxt.js остается предпочтительным вариантом для команд, работающих с Vue.js.

Выбор структуры веб-разработки должен определяться требованиями проекта, опытом команды и требованиями к производительности. В данной работе был выбран фреймворк Next.js, так как он является универсальным решением благодаря поддержке различных методов рендеринга (SSR, SSG, ISR), встроенной маршрутизации и наличию серверных функций. Это делает его подходящим выбором для создания масштабируемого интерфейса платформы электронной коммерции с возможностью интеграции с серверной частью и поддержкой адаптивного дизайна.

1.2.4 Использование Tailwind CSS в современных интерфейсах

Tailwind CSS представляет собой утилитарный CSS-фреймворк, который предлагает иной подход к оформлению по сравнению с классическими методами написания CSS. Вместо создания кастомных классов и стилей, в Tailwind используется набор готовых свойств, каждое из которых отвечает за одну конкретную функцию (например, p-4 - отступы, bg-blue-500 - цвет фона, text-center - выравнивание текста по центру и т.д.) [14].

Такой подход кардинально меняет процесс верстки - разработчик работает с HTML и Tailwind одновременно, что позволяет быстрее

прототипировать и изменять внешний вид компонентов без переключения между HTML и CSS-файлами. Также Tailwind CSS способствует унификации стиля и облегчает поддержку кода.

Кроме того, Tailwind обеспечивает консистентность дизайна за счет встроенной системы токенов - значений цветов, размеров, отступов и шрифтов. Это упрощает соблюдение единого визуального стиля во всем приложении. Уменьшение объема пользовательского CSS также положительно влияет на структуру проекта - код становится чище, а его поддержка упрощается.

Фреймворк Tailwind CSS хорошо интегрируется с современными инструментами, такими как React, Vue и Next.js, благодаря чему он органично вписывается в типичный стек фронтенд-разработки. Немаловажным преимуществом является и высокая степень кастомизации фреймворка - через конфигурационный файл tailwind.config.js его можно адаптировать под нужды конкретного проекта, добавляя собственные брейкпоинты, палитры цветов или типографику. Все это делает Tailwind CSS мощным и гибким инструментом для построения современных, адаптивных и визуально цельных интерфейсов.

В современных веб-приложениях Tailwind CSS активно используется для построения адаптивных, отзывчивых интерфейсов. Например, при создании платформ интернет-магазинов, административных панелей или дашбордов Tailwind позволяет быстро сверстать сетки карточек товаров, фильтры, таблицы и модальные окна. Благодаря адаптивным классам (md:, lg: и др.) легко реализовать мобильную и десктопную версии интерфейса в одном шаблоне.

Пример кода карточки товара на Tailwind CSS представлен в листинге 1.

Листинг 1 – Пример кода на Tailwind CSS

```
<div class="max-w-sm bg-white rounded-2xl shadow-md p-4">
    <img src="/product.jpg" alt="Товар" class="rounded-xl mb-4">
    <h3 class="text-lg font-semibold mb-2">Название товара</h3>
    <p class="text-gray-600 mb-4">Краткое описание товара.</p>
    <button class="bg-blue-500 text-white px-4 py-2 rounded-xl hover:bg-
blue-600">Купить
    </button>
</div>
```

Tailwind CSS является современным и эффективным инструментом для стилизации веб-интерфейсов. Он снижает порог входа в верстку, ускоряет разработку и делает код более поддерживаемым. Благодаря своей гибкости и возможности глубокой настройки Tailwind активно применяется в коммерческих проектах.

1.3 Влияние пользовательского опыта на эффективность E-commerce платформы

Современные E-commerce платформы функционируют в условиях высокой конкуренции, где даже незначительные недостатки в пользовательском опыте (UX) могут привести к потере клиентов, снижению конверсии и падению прибыли. Пользовательский опыт охватывает все аспекты взаимодействия человека с системой - от первой загрузки сайта до завершения покупки, включая восприятие дизайна, навигацию, скорость работы и интуитивность интерфейса. В данном разделе рассматривается влияние UX на ключевые метрики эффективности платформ электронной коммерции.

Пользовательский опыт - это совокупность ощущений и впечатлений пользователя при взаимодействии с цифровым продуктом. По определению Международной организации по стандартизации (ISO 9241-210), пользовательский опыт включает в себя восприятие, эмоции, реакции, а

также поведенческие и физиологические ответы, возникающие при использовании продукта [15].

В электронной коммерции пользовательский опыт приобретает особую значимость, поскольку напрямую влияет на поведение покупателей и ключевые метрики бизнеса. Удобная навигация, понятный и интуитивный интерфейс, визуальная привлекательность и высокая производительность сайта формируют первое впечатление о бренде и определяют вероятность совершения покупки. Также важны такие аспекты, как адаптивность дизайна, обеспечивающая корректную работу на различных устройствах, и доверие, которое создается за счет прозрачной информации о доставке, возвратах и безопасности оплаты.

Все эти характеристики пользовательского опыта отражаются на бизнес-показателях: повышение удобства использования интерфейса способно улучшить коэффициент конверсии, увеличить среднее время пребывания на сайте и средний чек, а также снизить показатель отказов. Пользователь, испытывающий положительные эмоции от взаимодействия с платформой, с большей вероятностью вернется за повторной покупкой. Как показывает исследование компании Forrester Research, каждый доллар, вложенный в UX, может принести до 100 долларов возврата инвестиций. [16]. Это делает UX не просто дизайнерским вопросом, а стратегическим инструментом бизнеса.

Тем не менее, в практике разработки E-commerce решений нередко встречаются типичные ошибки, которые негативно сказываются на пользовательском опыте. Среди них - перегруженная или непоследовательная навигация, медленная загрузка страниц, сложные формы заказа, недостаточная адаптивность под мобильные устройства, а также отсутствие визуальной иерархии и информативной обратной связи при ошибках.

Решения, доказавшие свою эффективность на успешных платформах, имеют минималистичный интерфейс с акцентом на продвигаемой товарной

единице, удобную систему поиска и фильтрации, упрощенную форму оформления заказа, а также адаптивный дизайн, ориентированный в первую очередь на мобильных пользователей. Использование визуальных подсказок, микроанимаций и интерактивных элементов делает взаимодействие более понятным и вовлекающим. Регулярное UX-тестирование, анализ пользовательского поведения и итеративные доработки на основе A/B-тестирования помогают постоянно улучшать интерфейс на основе реальных данных.

Оценка пользовательского опыта может проводиться как с помощью количественных, так и качественных методов. К числу наиболее распространенных инструментов относятся пользовательские тесты с участием реальных пользователей, тепловые карты, сбор метрик в Google Analytics и опросы с прямой обратной связью. Эти методы позволяют принимать обоснованные продуктовые решения, ориентируясь не на предположения, а на конкретные данные.

Таким образом, пользовательский опыт в электронной коммерции - это не просто часть дизайна, а важнейший элемент стратегии роста, развития и конкурентоспособности платформы. Грамотно спроектированный пользовательский опыт способствует укреплению лояльности, увеличению прибыли и общему успеху бизнеса. В рамках данной дипломной работы особое внимание уделяется именно UX-оптимизации, реализованной с использованием современных фреймворков, дизайна на основе макета и отзывчивой верстки.

1.4 Особенности адаптивной и кроссплатформенной верстки

В условиях стремительного роста числа пользователей мобильных устройств и разнообразия экранов различного форм-фактора (смартфоны, планшеты, ноутбуки, моноблоки и т.д.), адаптивная и кроссплатформенная верстка становится неотъемлемой частью современной веб-разработки. Для

электронной коммерции, где удобство и доступность интерфейса напрямую влияют на продажи и удовлетворенность клиента, данные подходы играют критическую роль.

Адаптивная верстка позволяет интерфейсу подстраиваться под размер экрана, меняя расположение, масштаб и видимость элементов в зависимости от разрешения устройства. Это не просто модное техническое решение, а жизненная необходимость. Более 60% интернет-пользователей совершают покупки именно с мобильных устройств, и, если сайт не оптимизирован под смартфоны, это автоматически означает потерю значительной части аудитории [17].

Кроме того, поисковые системы, такие как Google, отдают приоритет адаптивным сайтам в результатах поиска благодаря стратегии Mobile-First Indexing. С июля 2024 года Google полностью перешел на индексацию мобильных версий сайтов, что означает, что именно мобильная версия веб-страницы используется для оценки релевантности и ранжирования в поисковой выдаче [18]. Таким образом, отсутствие адаптивного дизайна может привести к снижению видимости сайта в поисковых системах и потере значительной части аудитории.

Кроссплатформенность, в свою очередь, означает корректную работу сайта в разных браузерах (Chrome, Firefox, Safari и др.) и на разных операционных системах (Windows, Android, iOS, macOS). Пользовательский интерфейс должен одинаково выглядеть и функционировать независимо от среды, иначе это может привести к путанице, ошибкам и негативному пользовательскому опыту.

Реализация адаптивного дизайна достигается с помощью ряда современных техник. Прежде всего, это использование медиа-запросов в CSS, которые позволяют задавать стили для разных ширин экрана. Также широко применяется подход "mobile-first" (англ. – мобильный в первую очередь), предполагающий начальную разработку интерфейса под мобильные устройства с последующей доработкой под стационарные

компьютеры. Кроме того, используются гибкие макеты на базе CSS Flexbox и Grid, процентные и относительные единицы измерения (%, em, rem) вместо фиксированных пикселей, а также адаптивные изображения.

В рамках данной работы особое внимание уделялось фреймворку Tailwind CSS - современной утилитарной CSS-библиотеке, которая позволяет реализовывать адаптивный интерфейс с помощью готовых классов, соответствующих ключевым breakpoints: sm, md, lg, xl и 2xl. Это значительно ускоряет процесс верстки и делает код более читаемым и структурированным.

Адаптивная верстка также предполагает учет особенностей пользовательского поведения: например, элементы управления должны быть достаточно крупными для удобного нажатия пальцем на экранах смартфонов (не менее 48×48 пикселей), горизонтальная прокрутка на мобильных устройствах недопустима, а порядок отображения блоков должен быть логичен для вертикальной прокрутки. Все это обеспечивает интуитивно понятный и комфортный пользовательский опыт.

Таким образом, адаптивная и кроссплатформенная верстка - это не просто часть дизайна, а ключевой компонент конкурентоспособности платформы электронной коммерции. Она обеспечивает доступность сайта для широкой аудитории, способствует увеличению конверсии, улучшает позиции в поисковой выдаче и помогает формировать положительное впечатление у пользователей. Использование современных инструментов, таких как Tailwind CSS и Next.js, значительно упрощает реализацию этих задач, позволяя сосредоточиться на повышении качества и эффективности пользовательского интерфейса.

1.5 Работа с прототипами и макетами интерфейсов (на примере Figma)

Проектирование пользовательского интерфейса UI является важнейшим этапом в разработке любого веб-приложения, особенно в сфере электронной торговли. От того, насколько грамотно и удобно спроектирован интерфейс, напрямую зависит пользовательский опыт, уровень удовлетворенности, а также показатели конверсии и лояльности клиентов. Одним из самых распространенных и мощных инструментов для проектирования и прототипирования интерфейсов в современной веб-разработке является Figma [19].

Figma представляет собой облачную платформу для создания интерфейсных макетов, интерактивных прототипов и командной работы дизайнеров и разработчиков. В отличие от многих десктопных аналогов, она может работать непосредственно в браузере, что делает ее доступной на любой операционной системе без необходимости установки. Figma позволяет создавать как простые, так и комплексные пользовательские интерфейсы, обеспечивая полный цикл разработки - от концепции до передачи готового макета в разработку.

Работа с макетом в Figma начинается с формирования общей структуры интерфейса, включая расположение ключевых элементов: навигационного меню, карточек товаров, фильтров, корзины, форм авторизации и личного кабинета. Эти элементы организуются в виде фреймов (frames), которые в Figma играют роль экранов будущего приложения. Каждому экрану можно присваивать размеры, соответствующие реальным устройствам - например, iPhone 14, MacBook Pro, iPad и т.д. Это особенно важно для проектирования адаптивных интерфейсов, когда один и тот же контент должен корректно отображаться на разных типах экранов.

Важной особенностью Figma является возможность работы с компонентами - это повторно используемые элементы, такие как кнопки,

карточки товара, поля ввода и другие интерфейсные блоки. Компоненты позволяют ускорить процесс дизайна, обеспечивают единый стиль и упрощают внесение изменений: редактируя один компонент, можно мгновенно обновить все его экземпляры на макете.

В рамках выполнения дипломной работы в качестве основы для веб-разработки использовался заранее подготовленный и предоставленный макет интерфейса E-commerce платформы, созданный в графическом онлайн-редакторе Figma. Этот макет содержал все основные страницы и компоненты будущего сайта: главную страницу, каталог товаров, карточки товаров, корзину, формы регистрации и авторизации, интерфейс личного кабинета, а также страницы управления профилем пользователя и заказами.

Одним из больших преимуществ Figma является встроенная система комментариев и совместной работы. В ходе разработки проекта это позволило согласовать макет с преподавателями и получать обратную связь в реальном времени. Комментарии можно размещать прямо на элементах макета, а все изменения автоматически сохраняются в облаке и доступны для просмотра другими членами команды.

Для веб-разработки особенно важна возможность экспорта макета и получения точных значений отступов, размеров, цветов и шрифтов. Figma предоставляет разработчику подробную информацию о каждом элементе: его координатах, размерах, стилях, шрифтах и значениях CSS. Это существенно ускоряет и упрощает верстку.

Особое внимание в процессе макетирования уделялось адаптивности интерфейса. В Figma предусмотрена возможность использовать автоматическое масштабирование и привязку элементов (constraints), позволяющую заранее задавать поведение компонентов при изменении размеров экрана. Это позволяет не только визуализировать, как будет вести себя интерфейс на различных устройствах, но и заложить правильную архитектуру адаптивной верстки еще до начала разработки.

Таким образом, использование Figma в процессе разработки интерфейса E-commerce платформы позволило:

- визуализировать целостную и логически структурированную систему экранов;

- обеспечить адаптивность и продуманную навигацию;

- ускорить процесс разработки за счет готовых компонентов и спецификаций;

- наладить эффективное взаимодействие между дизайнером и разработчиком;

- повысить итоговое качество пользовательского интерфейса и снизить количество правок на этапе верстки.

Figma выступает не просто как средство создания макетов, а как полноценный инструмент проектирования и коммуникации, позволяющий превратить идеи в ясный, визуально проработанный и технологически реализуемый интерфейс. Для платформ электронной коммерции, где каждая деталь влияет на конверсию, такая тщательная предварительная проработка интерфейса имеет решающее значение.

1.6 Постановка проблемы и формулировка задачи исследования

Современные E-commerce платформы сталкиваются с постоянно растущими требованиями пользователей к качеству интерфейса. Интерфейс - это первое, с чем взаимодействует пользователь, и именно от его удобства, логики и визуального оформления во многом зависит успех всей платформы. В условиях высокой конкуренции в онлайн-торговле даже незначительные недочеты в пользовательском опыте могут привести к снижению показателей вовлеченности, роста отказов и потере потенциальных клиентов. Это делает задачу оптимизации интерфейса особенно актуальной.

В рамках данной дипломной работы в качестве отправной точки выступал уже готовый дизайн-макет интерфейса E-commerce платформы,

предоставленный в Figma. Несмотря на наличие визуально проработанного макета, практика разработки показывает, что при реализации пользовательского интерфейса необходимо не только следовать дизайн-решениям, но и адаптировать их под реальные условия работы веб-приложения. Необходимо учитывать особенности поведения пользователей, технические ограничения браузеров, принципы адаптивности и кроссплатформенности, а также взаимодействие с бэкенд частью приложения. Это требует проведения целенаправленной работы по оптимизации как на уровне структуры интерфейса, так и на уровне реализации компонентов.

Задача оптимизации пользовательского интерфейса в рамках данной работы заключалась в обеспечении максимальной эффективности, удобства и скорости взаимодействия пользователя с платформой, при сохранении соответствия утвержденному макету. Для достижения этой цели необходимо было учесть и устранить ряд типичных проблем, возникающих при разработке интерфейсов для платформ E-commerce:

1. Сложная навигация и перегруженность страницы. Пользователь не должен тратить усилия на поиск нужной информации. Основные разделы - каталог, корзина, личный кабинет, заказы - должны быть легко доступны с любой страницы.

2. Недостаточная адаптивность. Интерфейс должен одинаково хорошо работать на различных устройствах: от смартфонов до широкоформатных мониторов. Это требует внедрения адаптивной верстки с корректной перестройкой элементов, без потери функциональности.

3. Слишком медленная загрузка или "тяжелые" компоненты. В E-commerce особенно важно сократить время загрузки страниц и обеспечить быструю реакцию интерфейса. Это требует оптимизации кода, правильного использования изображений, использования “ленивой” загрузки изображений и эффективной работы с асинхронными запросами.

4. Недостаточная интерактивность и обратная связь. При взаимодействии с интерфейсом пользователь должен получать мгновенную визуальную реакцию - анимации при наведении, подсказки, индикаторы загрузки и т. д.

5. Несогласованность внешнего вида компонентов. Интерфейс должен быть единообразным, использовать согласованные стили, шрифты, отступы, цвета и элементы управления.

На основе вышеуказанных проблем и целей можно сформулировать основную задачу исследования: разработать и реализовать оптимизированный пользовательский интерфейс E-commerce платформы на базе предоставленного макета в Figma, с использованием современных веб-технологий (Next.js, TypeScript, Tailwind CSS), обеспечивающий высокую степень удобства, адаптивности, производительности и визуальной целостности.

Для реализации поставленной задачи были определены следующие направления работы:

- реализация компонентной архитектуры интерфейса с учетом повторного использования элементов;
- применение Tailwind CSS для создания согласованного и масштабируемого дизайна;
- использование функциональных возможностей фреймворка Next.js для улучшения скорости загрузки страниц;
- обеспечение полной адаптивности интерфейса с учетом разных разрешений экранов;
- упрощение навигации, в том числе через упорядочивание маршрутов и взаимодействий;
- добавление интерактивных элементов для повышения отклика интерфейса;

- тестирование интерфейса на различных устройствах и браузерах с целью выявления и устранения потенциальных проблем верстки и отображения.

Таким образом, задача по оптимизации интерфейса охватывает как технические, так и пользовательские аспекты и направлена на повышение общей эффективности работы E-commerce платформы, улучшение пользовательского опыта и, как следствие, увеличение конверсии и удержания клиентов.

Выводы к разделу 1

В результате анализа современного состояния разработки пользовательских интерфейсов для E-commerce платформ можно сделать следующие обобщенные выводы, которые легли в основу проектирования и реализации интерфейса в рамках данной дипломной работы:

1. Пользовательский интерфейс является критически важной составляющей E-commerce платформ. Он напрямую влияет на восприятие сайта, удобство взаимодействия и конверсию пользователей. Наиболее востребованными задачами для интерфейса E-commerce платформ являются интуитивная навигация, удобная структура каталога товаров, легкость оформления заказов и возможность управления личным кабинетом.

2. Современные технологии веб-разработки предоставляют широкий инструментарий для создания гибких, производительных и масштабируемых интерфейсов. Среди них - фреймворк Next.js, язык TypeScript, и утилитарный CSS-фреймворк Tailwind CSS, которые обеспечивают баланс между скоростью разработки и качеством конечного продукта.

3. Пользовательский опыт играет ключевую роль в эффективности платформы. Исследования показывают, что удобство, предсказуемость и скорость работы интерфейса оказывают прямое влияние на лояльность и поведенческие метрики пользователей.

4. Адаптивная и кроссплатформенная верстка - неотъемлемый стандарт современных интерфейсов. Пользователи заходят на E-commerce платформы с самых разных устройств, и интерфейс должен одинаково хорошо отображаться и функционировать на всех из них. Фреймворк Tailwind CSS и гибкая система компонентов позволяют легко реализовать адаптивность на практике.

5. Использование заранее подготовленного макета в Figma обеспечивает структурированный и согласованный подход к разработке интерфейса. Такой макет позволяет разработчику сразу перейти к реализации, соблюдая единые стили, размеры и визуальные решения, заложенные дизайнером. Это ускоряет процесс разработки и снижает вероятность ошибок.

6. Задача по оптимизации интерфейса выходит за рамки простой реализации макета. Она включает технические и UX-аспекты: улучшение скорости загрузки, логичность навигации, повторное использование компонентов, минимизация лишних элементов и повышение общей интерактивности.

Таким образом, можно утверждать, что успешная реализация пользовательского интерфейса E-commerce платформы требует комплексного подхода - от понимания задач пользователей и выбора подходящих технологий до точной реализации адаптивного, производительного и визуально целостного решения.

В следующем разделе будет подробно рассмотрено проектирование и реализация пользовательского интерфейса с использованием фреймворка Next.js, TypeScript и Tailwind CSS на основе предоставленного макета и сформулированных требований.

Раздел 2. Проектирование и разработка пользовательского интерфейса с использованием Next.js

2.1 Обоснование архитектурных решений

Проектирование пользовательского интерфейса E-commerce платформы требует тщательного подхода к выбору архитектуры, поскольку от нее зависят масштабируемость, производительность, удобство поддержки и возможность дальнейшего развития проекта. В рамках данной дипломной работы для разработки фронтенд части проекта было принято решение использовать современный React-фреймворк - Next.js в связке с TypeScript и Tailwind CSS.

Основные преимущества фреймворка Next.js, которые обусловили его выбор:

1. Поддержка серверного рендеринга (SSR) и статической генерации страниц (SSG).

2. Файловая маршрутизация. Next.js использует файловую структуру для определения маршрутов, что упрощает организацию проекта и повышает читаемость кода.

3. Встроенная поддержка оптимизации изображений.

4. Интеграция с TypeScript “из коробки”.

Ключевым архитектурным решением стало внедрение компонентной структуры интерфейса. Все элементы реализуются в виде переиспользуемых React-компонентов (кнопки, карточки товаров, формы, навигационные панели). Это обеспечивает консистентность интерфейса, простоту внесения изменений, снижение дублирования кода и возможность масштабирования за счет композиции компонентов.

Файловая структура проекта была организована с учетом разделения по функциональным областям:

- components/;

- pages/;
- hooks/;
- styles/;
- utils/.

Это обеспечило модульность и облегчило навигацию по коду.

Также в проекте применялся Tailwind CSS, так как он позволил обеспечить скорость разработки, гибкость, адаптивность и возможность переопределять стили через конфигурацию. Tailwind CSS также позволил полностью воспроизвести макет из Figma, сохранив точные отступы, шрифты, цвета и размеры.

Использование TypeScript вместо чистого JavaScript обеспечило строгую типизацию и статическую проверку кода. Это повысило надежность проекта, позволило избежать большого количества тривиальных ошибок и упростило навигацию по коду в среде разработки. Также типизация оказалась особенно полезной при работе с внешними API и сложными структурами данных - такими как товары, пользователи, заказы и т.д.

Благодаря возможностям фреймворка Next.js, реализация маршрутизации происходила через автоматическое отображение файлов в папке pages/. Это позволило логично и понятно организовать структуру переходов по страницам сайта:

- /;
- /catalog;
- /product/[id];
- /cart;
- /profile.

Навигация была реализована с помощью встроенного компонента next/link, обеспечивающего клиентский роутинг без перезагрузки страницы.

Для повышения производительности и снижения времени отклика использовались функции getStaticProps и getServerSideProps в Next.js, а также динамический импорт компонентов. Кроме того, предусматривалось

кеширование данных, полученных с сервера, с целью минимизации количества сетевых запросов.

Таким образом, архитектурные решения, принятые в рамках дипломной работы, базируются на современных инструментах и подходах к разработке. Использование Next.js, TypeScript и Tailwind CSS обеспечило высокую гибкость, производительность, удобство сопровождения и соответствие текущим стандартам в сфере веб-разработки E-commerce решений.

2.2 Основы фреймворка Next.js

Фреймворк Next.js представляет собой одно из самых популярных решений для разработки современных веб-приложений на базе React. Он предлагает расширенную функциональность по сравнению с обычным React-приложением и активно используется для построения производительных, масштабируемых и SEO-оптимизированных интерфейсов. Благодаря широкому набору встроенных возможностей, Next.js стал особенно популярен в разработке E-commerce платформ, где важны скорость загрузки, гибкость и высокая отзывчивость интерфейса.

Одним из главных преимуществ Next.js является поддержка различных подходов к рендерингу:

- серверный рендеринг, SSR;
- статическая генерация страниц, SSG;
- инкрементная регенерация статических сайтов, ISR.

В дипломной работе использовалась статическая генерация страниц SSR, что позволило достичь баланса между производительностью и актуальностью данных.

Фреймворк Next.js использует файловую маршрутизацию, что означает, что каждая папка или файл внутри директории pages/ автоматически становится маршрутом сайта.

Например:

- pages/index.tsx — главная страница (/);
- pages/catalog/index.tsx — каталог (/catalog);
- pages/product/[id].tsx — динамическая страница товара (/product/123);
- pages/profile.tsx — личный кабинет (/profile).

Этот подход избавляет от необходимости вручную настраивать маршруты и значительно ускоряет разработку. Для создания динамических маршрутов используются квадратные скобки - [id], [slug], что удобно при работе с товарами, категориями и другими сущностями, имеющими уникальные идентификаторы.

Next.js предоставляет специальные функции для загрузки данных на серверной стороне:

- getStaticProps используется при SSG и позволяет получить данные во время сборки.
- getServerSideProps применяется для SSR - данные загружаются при каждом обращении к странице.
- getInitialProps - устаревший метод, заменен более гибкими функциями выше.

Эти функции возвращают данные, которые автоматически передаются в компонент страницы как props. Это позволяет централизованно управлять логикой загрузки данных и ускоряет начальный рендер.

Next.js предоставляет разработчику гибкие возможности для оптимизации под поисковые системы. Благодаря SSR и SSG, контент доступен поисковым ботам еще до выполнения JavaScript. Кроме того, используется компонент next/head, позволяющий настраивать мета-теги для каждой страницы индивидуально: заголовок, описание, ключевые слова, open graph и т.д. Для E-commerce платформ это дает преимущество в ранжировании товаров и категорий.

Фреймворк предоставляет встроенную оптимизацию изображений через компонент next/image, который автоматически подбирает нужное разрешение, применяет “ленивую” загрузку lazy-loading изображений, использует современный формат WebP, кэширует изображения и подстраивает их под размер экрана пользователя. Для E-commerce платформы, где каждый товар имеет изображение, это существенно ускоряет загрузку страниц и экономит трафик.

Фреймворк Next.js также предоставляет возможность создавать собственные API прямо в директории pages/api/. Это позволяет реализовать простую серверную логику (например, обработку форм, отправку писем, чтение данных из базы данных) без необходимости развертывания отдельного backend-приложения.

Next.js изначально совместим с TypeScript и предлагает автоматическую генерацию конфигурации при первом запуске. Использование TypeScript обеспечивает надежную типизацию props и данных, автодополнение в редакторе, предупреждение о типовых ошибках еще до запуска приложения, улучшение читаемости и сопровождения кода.

Для ускорения разработки используется встроенный dev-сервер с функцией горячей перезагрузки: при изменении кода компонента происходит автоматическое обновление страницы в браузере без перезапуска сервера. Это особенно удобно при стилизации компонентов с Tailwind CSS, где часто происходят мелкие правки интерфейса.

Таким образом, Next.js представляет собой мощную технологическую платформу, сочетающую в себе простоту React, производительность серверного рендеринга, удобную маршрутизацию, гибкость работы с данными и высокую SEO-эффективность. Все эти характеристики делают его идеальным выбором для разработки современного интерфейса E-commerce платформы, что и было подтверждено в рамках настоящей дипломной работы.

2.3 Структура приложения и маршрутизация в Next.js

Одной из ключевых особенностей фреймворка Next.js является его удобная структура приложения, построенная на основе файловой маршрутизации. Это становится особенно важно в контексте платформы электронной коммерции, где требуется большое количество логически связанных страниц (каталог, карточка товара, корзина, профиль и т.д.).

В рамках дипломной работы структура приложения была организована следующим образом (рисунок 1).

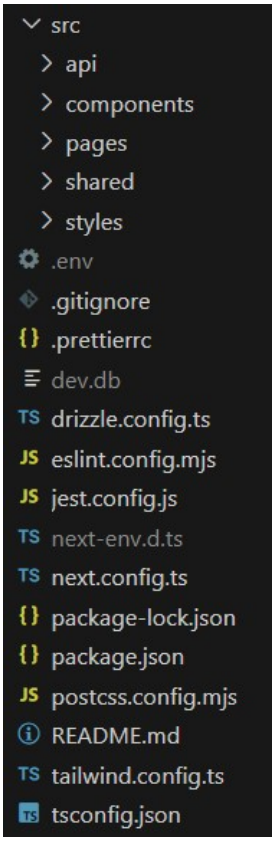

Рисунок 1 – Структура проекта

Весь исходный код приложения размещен внутри папки src, что позволило изолировать бизнес-логику от конфигурационных и служебных файлов, находящихся в корне проекта.

В директории src/api реализована серверная логика проекта с использованием API Routes Next.js. Здесь размещаются обработчики HTTP-запросов, работающие с базой данных.

Директория src/components содержит переиспользуемые React-компоненты, которые могут использоваться на разных страницах (формы, кнопки, карточки товаров).

Одной из ключевых директорий в Next.js является src/pages. Каждый файл в ней соответствует отдельному маршруту в приложении. Например, pages/index.tsx соответствует маршруту /. Здесь определяются страницы сайта и их серверная или клиентская логика.

Директория src/shared содержит общие модули, такие как утилиты, вспомогательные функции, пользовательские хуки, типы и константы. Эти элементы используются в разных частях приложения и способствуют повторному использованию кода.

В директории src/styles располагаются файлы со стилями. В случае с Tailwind CSS здесь находится основной файл импорта утилит и глобальных настроек.

Такая структура позволяет удобно масштабировать проект, разделяя визуальные компоненты, бизнес-логику, маршруты и вспомогательные утилиты.

В дополнение к основному коду, проект включает в себя набор конфигурационных и служебных файлов, обеспечивающих работу различных инструментов и технологий. Для хранения чувствительных данных в проекте используется файл переменных окружения .env. В этом файле хранятся ключи API и параметры подключения к базе данных. Система контроля версий Git исключает из отслеживания ненужные файлы и каталоги на основе содержимого файла .gitignore. За единый стиль форматирования кода отвечает Prettier, настройки которого указаны в файле .prettierrc.

Разработка велась с использованием локальной базы данных SQLite, представленной файлом dev.db. Связь с базой данных осуществляется через ORM-библиотеку Drizzle, параметры которой определены в drizzle.config.ts. Для анализа качества кода применяется ESLint, его поведение регулируется

файлом eslint.config.mjs. Тестирование реализовано с помощью Jest, настройки которого описаны в jest.config.js.

Конфигурация самого фреймворка Next.js хранится в файле next.config.ts, где задаются ключевые параметры сборки, маршрутизации и производительности. Список зависимостей, а также скрипты запуска и другая информация о проекте находятся в файлах package.json и package-lock.json.

Tailwind CSS интегрирован через PostCSS, настройки которого заданы в postcss.config.mjs. Индивидуальные параметры Tailwind хранятся в tailwind.config.ts. Файл tsconfig.json задает поведение компилятора TypeScript.

В отличие от традиционного подхода, где маршруты описываются вручную, Next.js использует автоматическую маршрутизацию, основанную на файловой системе. Каждый файл в директории pages/ автоматически превращается в маршрут веб-приложения. Например:

- pages/index.tsx → /
- pages/catalog/index.tsx → /catalog
- pages/product/[id].tsx → /product/123
- pages/cart.tsx → /cart
- pages/profile.tsx → /profile

Это позволяет логически структурировать маршруты прямо через структуру файлов, что особенно удобно в больших приложениях с множеством страниц.

Next.js поддерживает динамические маршруты через синтаксис с квадратными скобками - файл pages/product/[id].tsx будет соответствовать адресам /product/1, /product/abc, и т.д. В данной работе динамические маршруты используются для отображения страницы конкретного товара.

Чтобы получить доступ к параметру маршрута [id], используется следующий код:

Листинг 2 – Получение id товара

```
import { useRouter } from 'next/router';
const router = useRouter();
const { id } = router.query;
```

Next.js также поддерживает вложенные папки в pages/, что позволяет создавать логически связанные группы страниц. При необходимости можно также использовать динамическую вложенность, например pages/catalog/[category]/[productId].tsx для адресов типа /catalog/electronics/1.

Фреймворк Next.js предоставляет возможность настраивать поведение приложения на глобальном уровне с помощью специальных файлов внутри директории pages. Один из таких файлов - pages/_app.tsx. Это корневой компонент, который используется для оборачивания всего приложения. В нем удобно подключать глобальные стили, провайдеры контекста и сторонние библиотеки. В данном проекте файл _app.tsx использовался для подключения глобальных стилей и интеграции с библиотекой @tanstack/react-query, которая отвечает за кэширование и управление асинхронными запросами. Через компонент QueryClientProvider все приложение получает доступ к клиенту React Query, что позволяет удобно работать с загрузкой и кешированием данных.

В папке pages/api/ находятся обработчики запросов, которые выполняются на сервере.

Например:

- pages/api/products/index.tsx → /api/products;

- pages/api/cart/index.tsx → /api/cart.

Автоматическая маршрутизация, поддержка динамических и вложенных маршрутов, возможности API и единый подход к построению страниц делают Next.js мощным инструментом для создания современных платформ электронной коммерции.

2.4 Интеграция макета Figma и реализация дизайна с Tailwind CSS

Разработка качественного пользовательского интерфейса в современных веб-приложениях немыслима без четкого визуального ориентирования и согласования дизайна между дизайнером и разработчиком. Для реализации интерфейса платформы электронной коммерции в данной дипломной работе в качестве основы использовался макет десктопной версии приложения, предоставленный в Figma (рисунок 2).

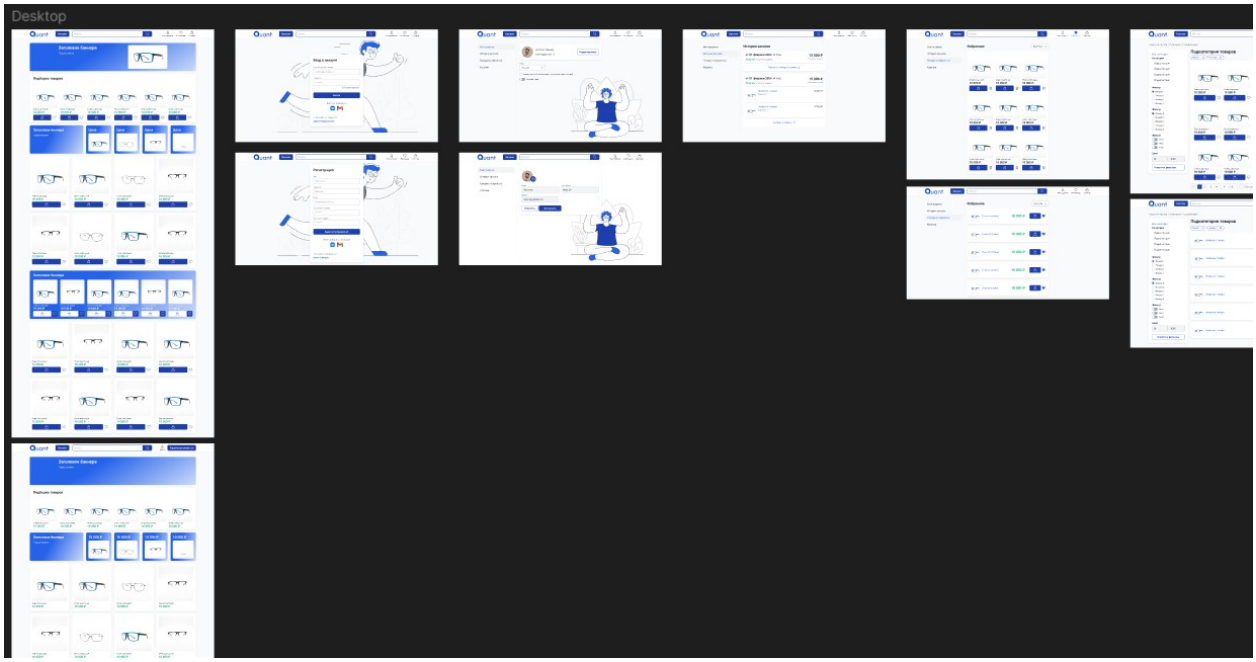

Рисунок 2 – Макет пользовательского интерфейса десктопной версии приложения в Figma

Макет содержал полный набор экранов и компонентов - главную страницу, каталог товаров, карточки, корзину, страницу оформления заказа, профиль пользователя, формы редактирования и модальные окна.

Кроме десктопной версии макета пользовательского интерфейса в Figma был предоставлен адаптированный мобильный макет (рисунок 3).

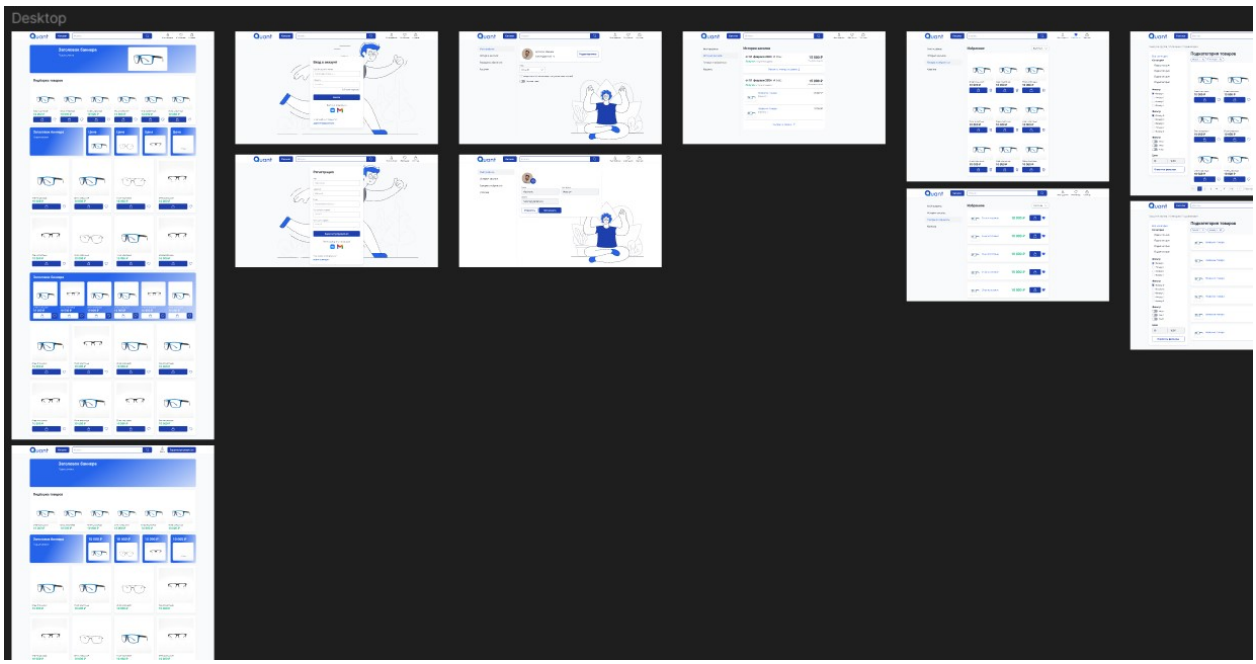

Рисунок 3 – Адаптированный мобильный макет пользовательского интерфейса приложения в Figma

Мобильный макет нужен, чтобы интерфейс оставался удобным и интуитивно понятным на небольших экранах.

Для реализации дизайна в дипломном проекте использовался фреймворк Tailwind CSS. Tailwind позволил писать стили непосредственно в .tsx файлах, что значительно ускорило процесс разработки и упростило отладку. Все компоненты проекта придерживаются единой системы размеров, цветов и отступов, что обеспечило визуальную целостность интерфейса.

Для превращения дизайна в Figma в полностью функционирующий компонент макеты страниц были проанализированы. Работа с макетом включала следующие задачи:

- определение точек перелома (breakpoints) для разных ширин экрана (375 px, 768 px, 1024 px и выше);

- измерение размеров шрифтов (14 px, 16 px, 24 px, 30 px) и межстрочных интервалов (line-height) для заголовков, описаний и вспомогательных текстов;

- задание отступов и интервалов между элементами (mt-4 - это 16 пикселей, mb-8 - это 32 пикселя);

- извлечение кодов цветов (основной темно-серый цвет #1F2937, акцентный синий #1E40AF, зеленый #10B981 для цен, светло-серый #6B7280 для второстепенного текста и фоновые цвета);

- определение поведения элементов при наведении и нажатии (hover/active-состояния);

- сопоставление повторяющихся элементов с потенциальными переиспользуемыми компонентами.

Вместо того, чтобы писать собственные стили CSS, использовался синтаксис произвольных значений Tailwind, чтобы соответствовать точным значениям из макета - классы вроде text-[24px], md:text-[30px] или text-[#1F2937] позволили отразить дизайн пиксель в пиксель.

На рисунке 4 представлена главная страница реализованной платформы электронной коммерции. Здесь располагаются ключевые элементы интерфейса: шапка сайта с навигацией, кнопка входа в аккаунт пользователя, поиск товаров, главный баннер, список популярных товаров.

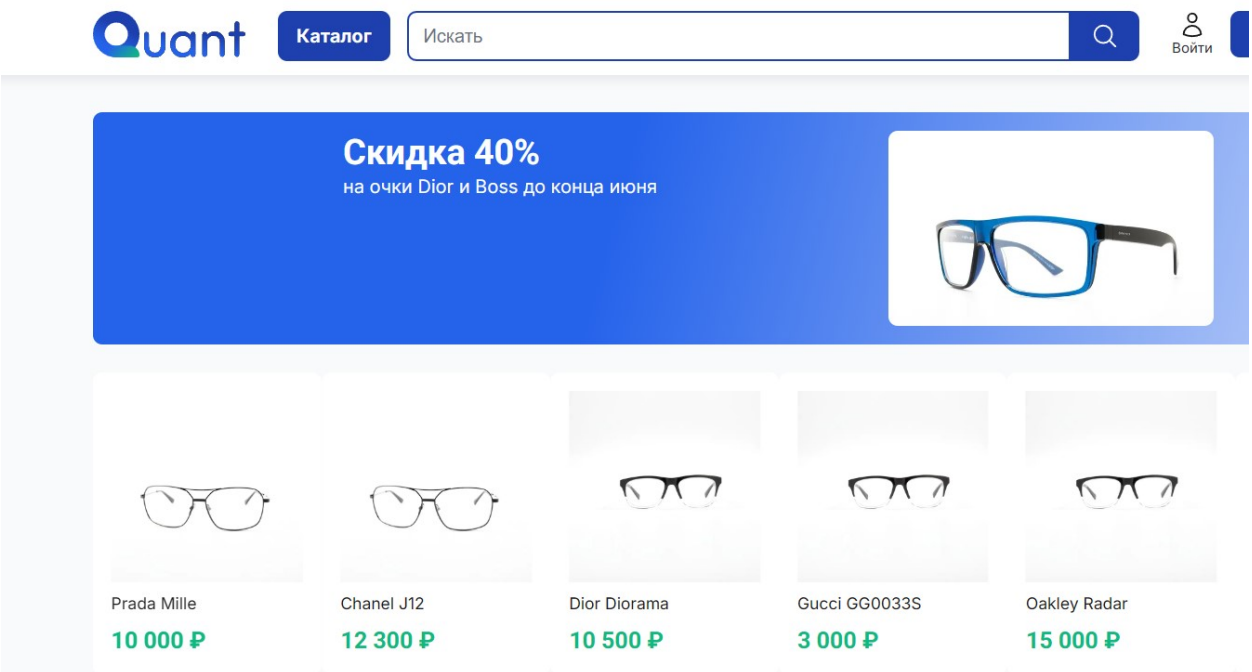

Рисунок 4 – Главная страница платформы электронной коммерции

Главная страница - это первая точка контакта с пользователем, поэтому особое внимание уделялось чистоте дизайна, скорости загрузки и логике взаимодействия.

В ходе выполнения дипломной работы по макету в Figma был разработан адаптивный функциональный компонент страницы товара. На рисунке 5 представлен вид страницы товара с шириной окна браузера 1440 пикселей, а на рисунке 6 – эта же страница с шириной окна браузера 390 пикселей.

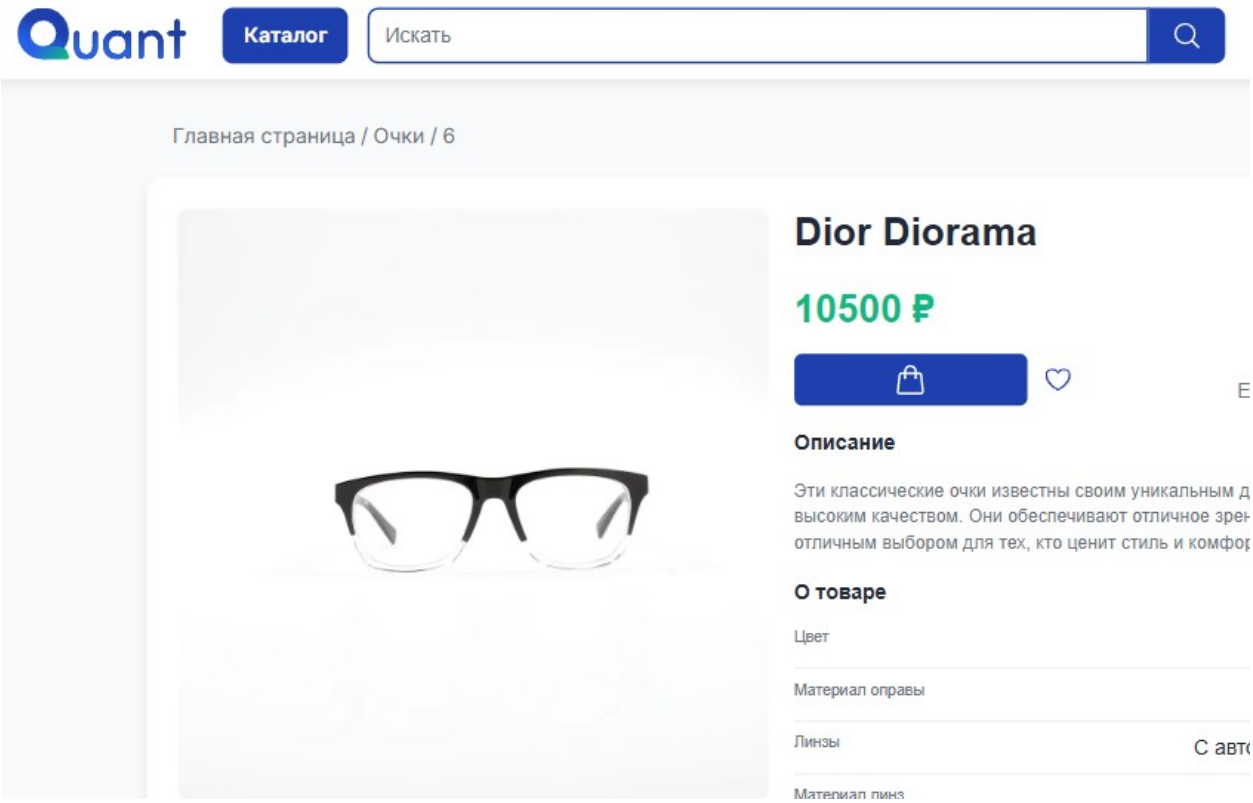

Рисунок 5 – Страница товара с шириной окна браузера 1440 пикселей

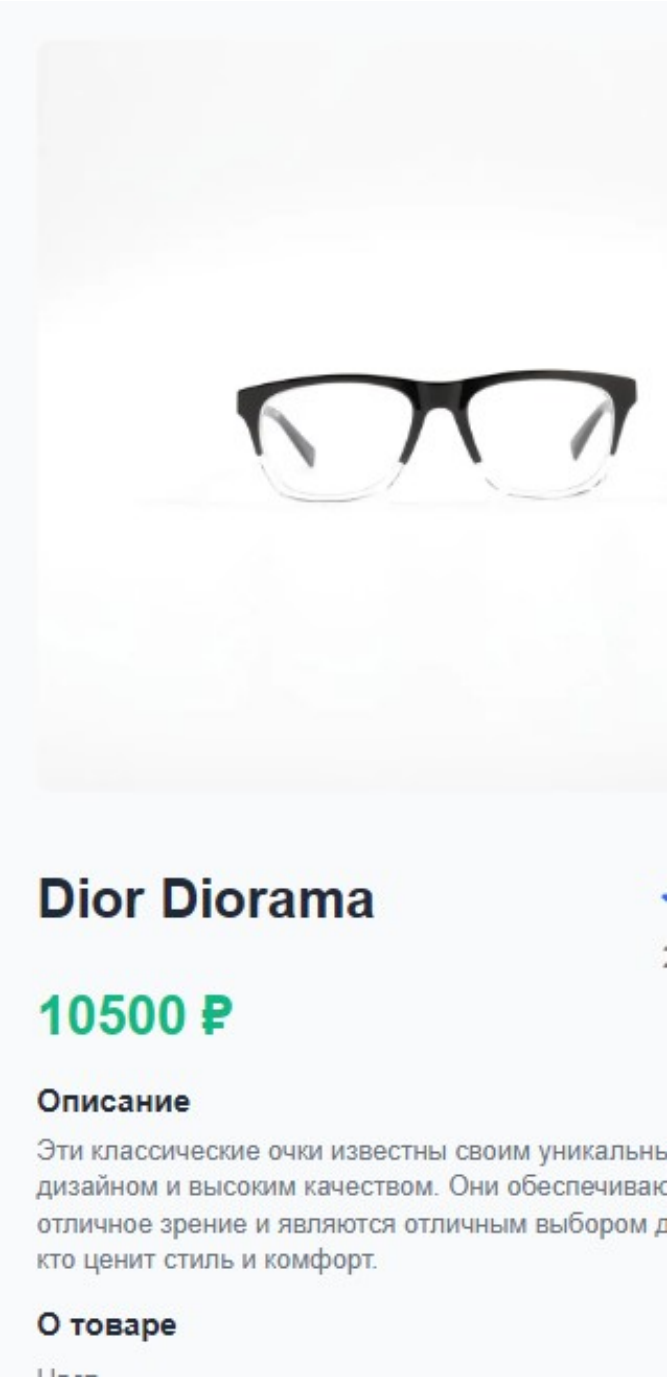

Рисунок 6 – Страница товара с шириной окна браузера 390 пикселей

Исходный код страницы товара разделен на два основных файла (Приложение 1):

- pages/[id].tsx, который отвечает за получение идентификатора товара, и загрузку данных о товаре.

- components/ProductPage/ProductPage.tsx, который непосредственно реализует рендеринг визуальной части страницы продукта, включая галерею

изображений, название, рейтинг, цену, кнопки управления корзиной, “лайками”, а также список характеристик.

Такое разделение соответствует принципу “контейнер–компонент”: в контейнерном компоненте обрабатываются запросы данных и бизнес-логика, а в презентационном - непосредственно верстка и работа с Tailwind CSS.

Рассмотрим подробнее компонент ProductPage (Приложение 1). Контейнер страницы ограничивает ширину до 980 пикселей и центрирует содержимое за счет mx-auto. Значение 980 пикселей получено из макета Figma как максимальная ширина контента для больших экранов. На меньших экранах этот контейнер плавно сжимается, обеспечивая надлежащие отступы и поля.

Раздел галереи на странице товара в макете требовал большого главного изображения, сопровождаемого четырьмя миниатюрами. Чтобы реализовать это, компонент сначала определяет основную фотографию по умолчанию. Эта фотография отображается в элементе div, стилизованном с использованием классов Tailwind CSS min-w-[335px] и min-h-[288px] для обеспечения правильных минимальных размеров на мобильных устройствах, и с применением классов bg-white rounded-lg overflow-hidden flex items-center justify-center для достижения чистого, центрированного вида. Как только пользователь нажимает на одно из изображений миниатюр - каждое из которых размещено в своем собственном div с классами w-[97px], h-[96px], bg-gray-200 и rounded-lg hover:cursor-pointer - состояние mainPhotoLink обновляется, и Tailwind гарантирует, что новое изображение заполнит контейнер с помощью w-full h-full object-cover. На больших экранах (с шириной больше или равной 768 пикселей) строка миниатюр становится видимой из-за скрытого md:flex justify-center gap-2 mt-4, точно воссоздающего интервалы и выравнивание из макета Figma.

Текстовые элементы, такие как название продукта, рейтинг и цена, были заданы классами Tailwind, которые отражают указанные в макете размеры шрифта и выбранную цветовую палитру.

Название продукта находится внутри тега <h1> с классами font-bold text-[24px] md:text-[30px] leading-8 md:leading-9 text-[#1F2937]. Рядом с ним отображается номер рейтинга с иконкой звезды SVG, импортированной через компонент изображения Next.js. Цена отображается ярким зеленым цветом #10B981 с использованием text-[#10B981] font-bold text-[24px] md:text-[30px] leading-8 md:leading-9, что позволяет ей выделяться именно так, как того требует дизайн.

В десктопной версии внешний тег div использует hidden md:flex mt-4 justify-between gap-4, чтобы весь блок управления отображался только на экранах шириной больше или равной 768 пикселей. Внутри условная проверка inBasket (вычисляемая путем проверки элементов корзины) определяет, показывать ли компонент ReplaceQuantity с количеством добавляемого товара в корзину или простую синюю кнопку “Добавить в корзину”. Классы этой кнопки - bg-[#1E40AF] w-[180px] h-[40px] text-white rounded-[6px] flex items-center justify-center - соответствуют указанному в макете Figma размеру и синему фону.

На мобильных устройствах эти элементы управления объединяются в одну строку, чтобы такой макет отображался только на телефонах. Кнопка "Добавить в корзину" занимает всю ширину (flex-1), сохраняя высоту 40 пикселей и синий фон. Если товар закончился, в кнопке добавления товара в корзину значок с корзиной заменяется текстом “Нет в наличии”.

Под кнопкой добавления товара в корзину отображается описание товара и таблица с его характеристиками. Поскольку в макете характеристики товара организованы в строки, разделенные горизонтальной линией, в компоненте перебирается объект с ключами “Цвет”, “Материал оправы” и т.д. Каждая пара ключ-значение отображается в div с flex justify-between и условным набором классов: первая строка получает нижнюю границу и дополнительный отступ (border-b pb-[6px] md:pb-2), промежуточные строки - отступы сверху и снизу (border-b py-[6px] md:py-2),

последняя строка - только отступ сверху (pt-[6px] md:pt-2), чтобы не рисовать лишнюю границу.

В результате получилась страница товара, которая выглядит и ведет себя точно так, как предусмотрел дизайнер в макете Figma.

Также в ходе выполнения дипломной работы по макету в Figma был разработан адаптивный функциональный компонент профиля пользователя. На рисунке 7 представлен вид профиля пользователя с шириной окна браузера 1440 пикселей, а на рисунке 8 – эта же страница с шириной окна браузера 390 пикселей.

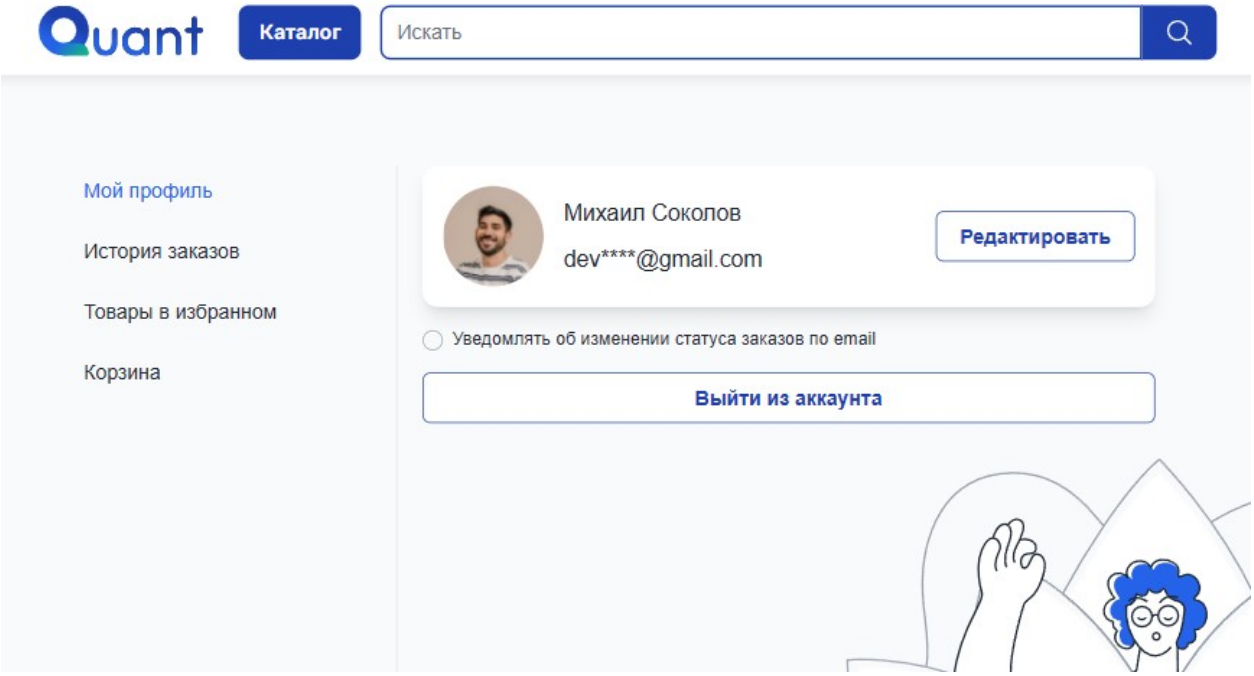

Рисунок 7 – Страница профиля пользователя с шириной окна браузера 1440 пикселей

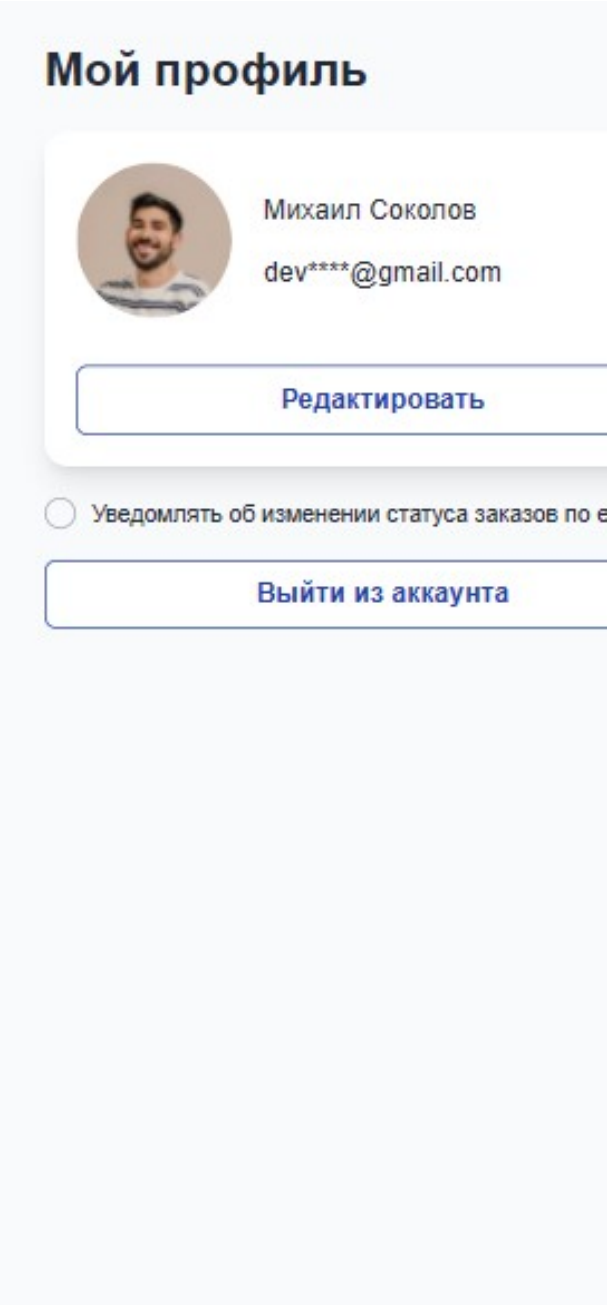

Рисунок 8 – Страница профиля пользователя с шириной окна браузера 390 пикселей

Также на рисунке 9 представлен вид профиля пользователя в режиме редактирования с шириной окна браузера 1440 пикселей, а на рисунке 10 – эта же страница с шириной окна браузера 390 пикселей.

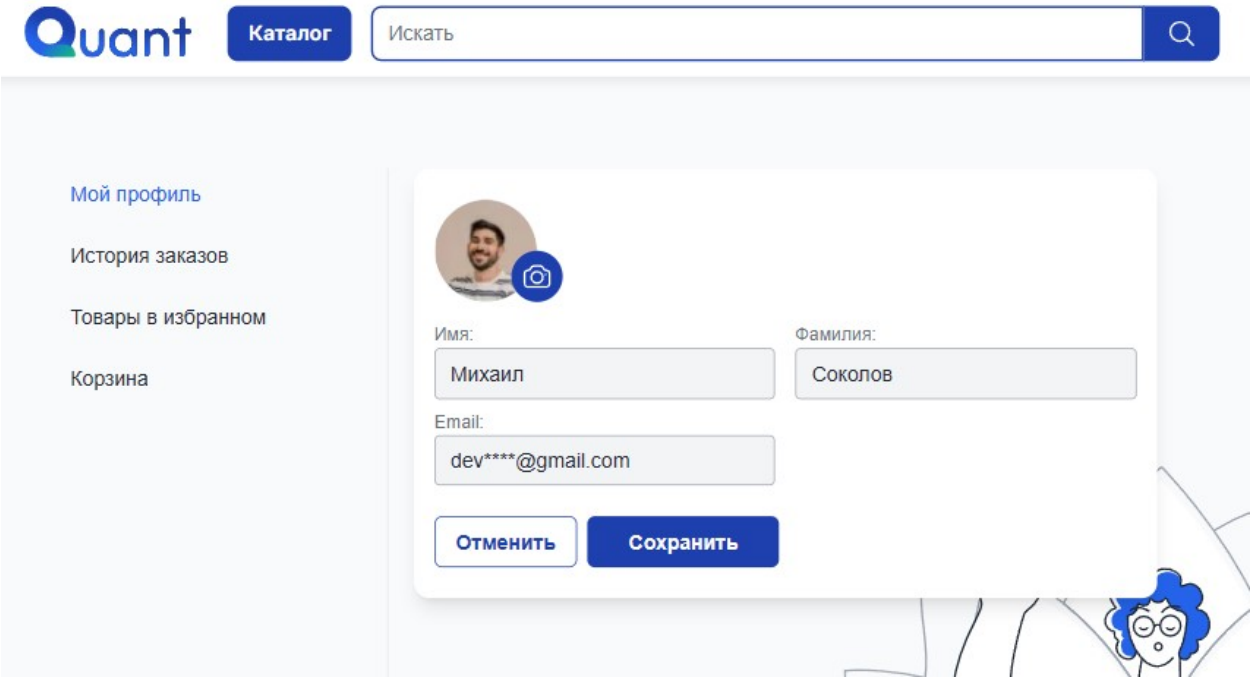

Рисунок 9 – Страница профиля пользователя в режиме редактирования с шириной окна браузера 1440 пикселей

Страница профиля пользователя состоит из двух ключевых компонентов – Profile и ProfileSection (Приложение 1). Компонент Profile – это контейнер страницы, отвечающий за бизнес-логику, взаимодействие с состоянием пользователя и маршрутизацией. Компонент ProfileSection - визуальный компонент, отвечающий за отображение и редактирование пользовательских данных.

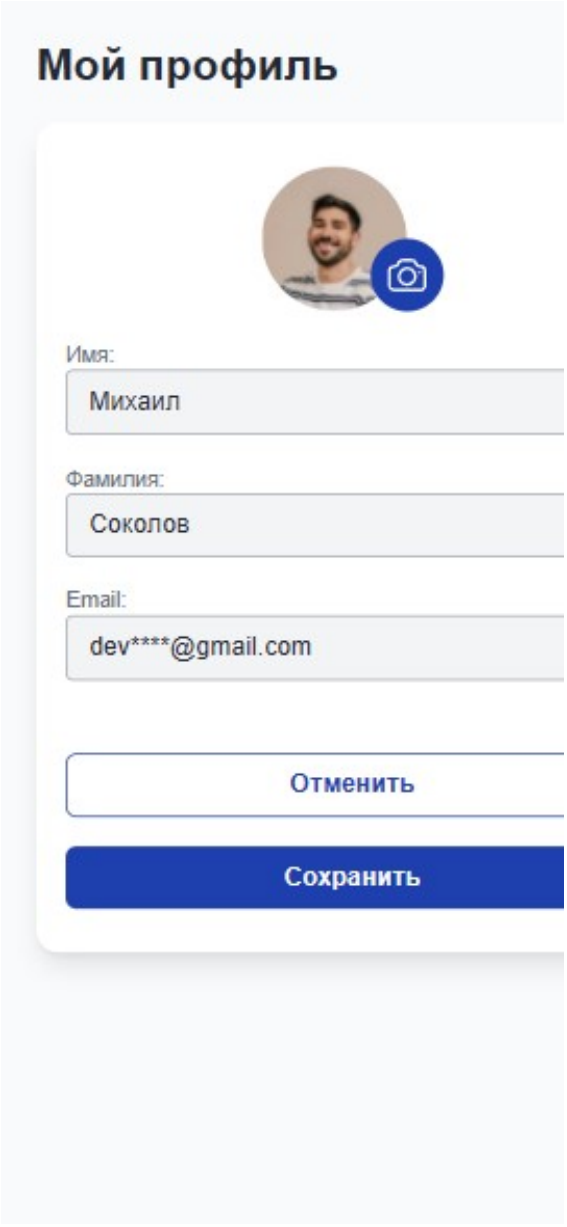

Рисунок 10 – Страница профиля пользователя в режиме редактирования с шириной окна браузера 390 пикселей

Макет страницы профиля в Figma включал в себя следующие элементы - аватар пользователя, форму для редактирования персональных данных (имя, фамилия, email), элементы управления (кнопки "Редактировать", "Сохранить", "Отменить") и визуальное оформление.

Элементы div с классами bg-white, rounded-xl, shadow-lg соответствуют карточке из дизайн-системы, где цвет фона - белый, скругление - 0.75rem (класс rounded-xl), а тень - shadow-lg. Ограничения по ширине max-w-[720px] и md:max-w-[580px] взяты из макета Figma, где на больших экранах (ширина

экрана равна или более 768 пикселей) ширина должна быть жестко ограничена 580 px, а на меньших экранах - 720 px или подстраиваться под реальную ширину контейнера.

В макете Figma аватар располагается по центру внутри контейнера в режиме редактирования. Соответственно, в .tsx добавлено сочетание классов, представленное в листинге 3.

Листинг 3 – Позиционирование аватара с помощью Tailwind CSS

```
<div className="flex justify-center md:block">
  <div className="relative w-[100px] h-20">
    <Image … className="rounded-full" …/>
    <div  className="absolute  bottom-0  right-0  bg-primary  p-2  rounded-full
cursor-pointer">
      <Image src={CameraIcon} … />
    </div>
  </div>
</div>
```

Для позиционирования аватара в мобильной версии применен flex justify-center. У родительского контейнера relative позволяет осуществить абсолютное позиционирование иконки камеры (absolute bottom-0 right-0), как это было задумано в дизайне (иконка “наезжает” на нижний правый край аватара). Размеры аватара взяты из Figma - круг должен быть размером 80×80 пикселей.

В макете предусмотрены одинаковые отступы и заливка для пяти основных полей страницы профиля. С помощью Tailwind заданы следующие свойства:

- px-3 py-[7px] - горизонтальные и вертикальные внутренние отступы.

- font-normal text-[14px] leading-5 md:text-[16px] md:leading-6 - размеры шрифта и межстрочные интервалы, взятые из Figma (на мобильной версии текст должен быть чуть меньше, на планшетах/десктопах - стандартные 16 пикселей).

- text-[12px] leading-4 md:text-[14px] md:leading-5 text-text-muted font-normal для label (текст “Имя”, “Фамилия”, “Email”).

- border border-input-border - граница толщиной 1 пиксель.

- цвета #9CA3AF для input-border, #F3F4F6 для bg-input-bg (цвет фона).

- rounded-[4px] - скругление 4 пискеля.

Макет задает поля “Имя” и “Фамилия” в одну строку (две колонки) на разрешениях равных или больше 768 пикселей, тогда как на мобильных они располагаются один под другим. Этого эффекта достигают классами (листинг 4) flex flex-col (на мобильных устройствах), при этом на экранах шире 768 пикселей применяется flex-row. Каждая колонка забирает половину ширины (w-1/2). Отступ между полями задан через gap-4, что соответствует промежутку 16 px в макете.

Листинг 4 – Позиционирование полей “Имя” и “Фамилия” с помощью Tailwind CSS

```
<div className="flex flex-col md:flex-row gap-4">
  <div className="w-full md:w-1/2">…</div>
  <div className="w-full md:w-1/2">…</div>
</div>
```

По дизайну в Figma кнопка “Отменить” должна быть с прозрачным фоном, синей границей и текстом синего цвета. Кнопка “Сохранить” должна быть с синим фоном и белым текстом. При наведении мыши фон и текст меняют цвет. С помощью Tailwind CSS в проектной работе это реализовано так:

- классы border-primary и text-primary используют цвет #1E40AF (синий);

- hover:bg-blue-50 добавляет бледно-голубой фон для кнопки “Отменить” при наведении мыши;

- hover:bg-blue-700 добавляет темно-синий фон для кнопки “Сохранить” при наведении мыши;

- класс rounded-md задает радиус по макету.

Таким образом, в ходе данной дипломной работы был реализован интерфейс страницы товара и страницы профиля пользователя платформы электронной коммерции на основе дизайн-системы, представленной в Figma. Для обеспечения точного соответствия использован фреймворк Tailwind CSS. Стили компонентов реализованы с учетом мобильных и десктопных разрешений. Это позволяет интерфейсу оставаться удобным для восприятия на всех типах устройств.

Реализованные страницы демонстрирует эффективную практику переноса интерфейсных решений из Figma в реальную среду с использованием Tailwind CSS.

2.5 Компонентный подход и организация кода на TypeScript

Одним из ключевых факторов, обеспечивающих масштабируемость, поддержку и повторное использование кода в современных веб-приложениях, является компонентный подход. В рамках разработки пользовательского интерфейса платформы электронной коммерции на базе Next.js была реализована структура, основанная на разбиении интерфейса на переиспользуемые компоненты.

Компонентный подход предполагает представление интерфейса как совокупности мелких, изолированных, логически завершенных блоков, каждый из которых отвечает за свою часть UI дизайна. Примеры таких компонентов в рамках проекта:

- Sidebar - левая навигационная панель;
- ReplaceQuantity – подсчет количества товара в корзине;
- Breadcrumbs – показывает текущий путь;
- LoadingIcon – иконка загрузки данных с сервера;
- ProfileBackground – установка фоновой картинки;
- Button, Input - элементарные UI-компоненты.

Каждый компонент разрабатывался в отдельном файле и размещался в каталоге components/, что облегчает навигацию по проекту и повторное использование кода. Каждый компонент включает в себя tsx-разметку и стилизацию с использованием Tailwind CSS, а при необходимости типизацию props и обработчики событий. Пример простого компонента ProfileBackground представлен в листинге 5.

Листинг 5 – Реализованный адаптивный блок описания товара

```
import { ReactNode } from "react";

type ProfileBackgroundProps = {
  children: ReactNode;
  imageUrl?: string;
}
export default function ProfileBackground({ children, imageUrl }:
ProfileBackgroundProps ) {
  return (
    <div
      className="bg-no-repeat bg-bottom bg-[right_162px] grow bg-transparent
md:bg-[image:var(--bg-image)]"
     style={{"--bg-image":url(${imageUrl|| "/images/sport_meditation.svg"})`
} as React.CSSProperties}
    >
      { children }
    </div>
  );
}
```

Компонентный подход обеспечил модульность и повторное использование кода, а применение TypeScript - надежную проверку типов. Такая архитектура делает проект гибким и масштабируемым, позволяя безболезненно вносить изменения и развивать функциональность в будущем.

2.6 Разработка функциональности страницы профиля пользователя

Создание платформы электронной коммерции требует реализации страницы профиля пользователя. Этот раздел платформы обеспечивает не только удобство клиента, но и напрямую влияет на конверсию и пользовательский опыт. Профиль пользователя предоставляет пользователю доступ к базовой информации о себе, при этом должна быть возможность редактировать эти данные.

На этапе реализации проекта был создан компонент страницы профиля, отображающий данные пользователя - имя, фамилию и адрес электронной почты. Основные возможности страницы профиля:

- просмотр и редактирование данных профиля;
- выход из аккаунта с очисткой пользовательского состояния;
- настройка уведомлений о статусе заказов.

Компонент страницы профиля построен с использованием следующих компонентов:

- ProfileSection - форма отображения и редактирования профиля;
- Sidebar - навигационное меню по разделам;
- ProtectedRoute - защита страницы от неавторизованного доступа;
- useMutation из React Query - отправка запросов на обновление данных;
- useUserStore (на базе Zustand) - хранение пользовательских данных в состоянии клиента.

Данные пользователя берутся из глобального хранилища состояния, построенного с использованием библиотеки Zustand:

Листинг 6 – Получение данных пользователя из глобального хранилища

```
const name = useUserStore((state) => state.name);
const surname = useUserStore((state) => state.surname);
const email = useUserStore((state) => state.email);
```

Локальное состояние используется для управления редактируемыми полями:

Листинг 7 – Использование локального состояния

```
const [profileData, setProfileData] = useState({ name, surname, email });
```

Синхронизация локального состояния с глобальными данными происходит через useEffect:

Листинг 8 – Использование useEffect при изменении данных пользователя

```
useEffect(() => {
  setProfileData({ name, surname, email });
}, [name, surname, email]);
```

Форма редактирования реализована через компонент ProfileSection, который переключается в режим редактирования с помощью флага isEditing. Сохранение новых данных инициирует вызов мутации через React Query:

Листинг 9 – Мутация через React Query

```
const mutation = useMutation({
  mutationFn: updateUser,
  onSuccess: (_, variables) => {
    useUserStore.getState().setUserData({
      name: variables.name ?? "",
      surname: variables.surname ?? "",
      email: variables.email ?? "",
      avatar: "",
    });
    setIsEditing(false);
    console.log('Профиль успешно обновлен');
  },
  onError: (error) => {
    console.error('Ошибка при обновлении профиля:', error);
  },
});
```

Обновленные данные отправляются на сервер через mutation.mutate(updatedData), после чего они записываются в глобальное состояние для синхронизации интерфейса.

Компонент ProtectedRoute используется для ограничения доступа к странице профиля только авторизованным пользователям. Дополнительно используется хук useProtectedRoute, обеспечивающий защиту от несанкционированного доступа:

Листинг 10 – Ограничение доступа к странице профиля

```
<ProtectedRoute protection={useProtectedRoute}>
  ...
</ProtectedRoute>
```

Также реализован механизм выхода из аккаунта. Функция handleLogout отправляет запрос на сервер для завершения сессии, очищает глобальное состояние и перенаправляет пользователя на главную страницу:

Листинг 11 – Выход из аккаунта пользователя

```
const handleLogout = async () => {
  try {
    await logOut();
  } finally {
    useUserStore.getState().removeUserData();
    useUserStore.getState().setIsAuthenticated(false);
    router.push("/");
  }
};
```

Разработка личного кабинета включала в себя не только визуальное отображение пользовательских данных, но и полноценную обработку состояния, защиту маршрута, взаимодействие с API и интеграцию с глобальным хранилищем. Такой подход обеспечивает безопасный и отзывчивый интерфейс.

2.7 Использование API и взаимодействие с серверной частью

Эффективное взаимодействие между клиентской и серверной частью является ключевым аспектом при разработке современных веб-приложений. В данной дипломной работе реализация фронтенд-интерфейса на базе Next.js предусматривала интеграцию с серверной частью через REST API.

Проект реализован с использованием React Query, что позволило централизовать логику взаимодействия с API и упростить кеширование, повторное использование данных и автоматическое обновление интерфейса. Все API-запросы оформлены в виде асинхронных функций, а обращение к ним осуществляется через хуки useQuery и useMutation.

Для загрузки списка товаров использовался пользовательский хук useProducts, который запрашивал данные у сервера и кешировал их с помощью React Query:

Листинг 12 – Применение пользовательского хука useProducts

```
const { products } = useProducts();
```

Аналогично, при загрузке информации об избранных товарах применялся следующий запрос:

Листинг 13 – Применение useQuery

```
const { data } = useQuery({
  queryKey: ["favoritesInfo"],
  queryFn: getFavoritesInfo,
  enabled: isAuthenticated,
});
```

Вся работа с кешем централизована - при успешном изменении избранного товара выполняется автоматическое обновление данных:

Листинг 14 – Применение useMutation

```
const mutation = useMutation({
  mutationFn: getFavoritesInfo,
  onSuccess: () => {
    queryClient.invalidateQueries({ queryKey: ["favoritesInfo"] });
  },
});
```

Данные корзины также запрашивались через асинхронный хук:

Листинг 15 – Применение useBasket

```
const basketQuery = useBasket();
const basketItems =
  isAuthenticated && basketQuery?.basket ? basketQuery.basket.items : [];
```

Для защиты страниц, требующих авторизации, реализован компонент ProtectedRoute, который оборачивает защищенные маршруты. При выходе из аккаунта происходила очистка состояния и перенаправление пользователя. Аутентификация происходила с помощью httpOnly cookie, что повышает безопасность хранения токенов.

Для оптимизации производительности и SEO использовался Server-Side Rendering (SSR) подход к рендерингу страниц с товарами, чтобы обеспечить быструю индексацию поисковыми системами:

Листинг 16 – Использование асинхронной функции getServerSideProps

```
export const getServerSideProps: GetServerSideProps = async (context) => {
  const { id } = context.params!;
  const productId = parseInt(id as string, 10);
  return {
    props: { productId },
  };
};
```

Для повышения отказоустойчивости интерфейса везде была реализована обработка состояний загрузки и ошибок:

Листинг 17 – Использование асинхронной функции getServerSideProps

```
if (loading) return <Spinner />;
if (error) return <ErrorMessage message={error.message} />;
```

Взаимодействие с серверной частью реализовано с помощью современных инструментов: React Query, Zustand, SSR и защищенной аутентификации. Четкое разделение ответственности, использование хуков, кеширование данных и обработка ошибок обеспечили производительность, безопасность и стабильность пользовательского интерфейса.

Выводы к разделу 2

В данном разделе была подробно рассмотрена архитектура и реализация клиентской части платформы электронной коммерции с использованием современных технологий веб-разработки. Основой разработки выступил фреймворк Next.js, что позволило создать быстрый, отзывчивый и масштабируемый интерфейс, соответствующий требованиям современных веб-приложений.

Принятые архитектурные решения обеспечили четкое разделение логики, масштабируемость проекта и возможность удобного рефакторинга. Благодаря компонентному подходу, интерфейс был разбит на переиспользуемые блоки, что существенно упрощает поддержку и развитие проекта. Использование TypeScript повысило надежность разработки за счет строгой типизации, а Tailwind CSS обеспечил быстрое и гибкое стилизование элементов.

В рамках работы была успешно выполнена интеграция заранее подготовленного макета из Figma, что обеспечило соответствие визуальной части требованиям дизайна. Все ключевые функции были адаптированы для мобильных устройств.

Отдельное внимание было уделено взаимодействию с серверной частью. Через REST API обеспечивалась загрузка и обновление всех данных на клиенте. Реализация безопасной аутентификации и защита пользовательских данных обеспечили соответствие современным требованиям по безопасности.

Таким образом, результаты проектирования и разработки пользовательского интерфейса показывают, что выбранные инструменты и подходы позволяют создавать эффективные, масштабируемые и удобные в использовании платформы электронной коммерции. Полученная архитектура интерфейса готова к дальнейшему расширению, оптимизации и внедрению новых функциональных модулей.

Раздел 3. Тестирование, оценка и оптимизация пользовательского интерфейса

3.1 Обзор существующих решений и UX-практик конкурентов

Чтобы создать конкурентоспособный и удобный интерфейс для платформы электронной коммерции, важно учитывать успешные практики ведущих игроков рынка. Анализ существующих решений помогает выявить лучшие подходы к UX, общие шаблоны взаимодействия и ожидания пользователей от онлайн-покупок. В данном подразделе проведен сравнительный обзор популярных E-commerce платформ с точки зрения интерфейсной реализации, удобства навигации, адаптивности и визуальной подачи информации.

Платформы для анализа были выбраны за высокий уровень доверия пользователей и узнаваемости бренда. К ним относятся:

- Wildberries, крупнейший российский интернет-ритейлер;
- Ozon, комплексная торговая площадка с широким ассортиментом;
- AliExpress, глобальная платформа с международным охватом;
- Lamoda, нишевый магазин, ориентированный на моду.

Анализ был сосредоточен на нескольких ключевых аспектах: структура домашней страницы, функциональность поиска и фильтрации, макет страниц продукта, процессы добавления товаров в корзину и оформления заказа, функции учетной записи пользователя, мобильная верстка и производительность интерфейса.

На этих платформах было выявлено несколько последовательных принципов UX дизайна. Современные интерфейсы, как правило, отдают приоритет минимализму и визуальной иерархии - дизайн ориентирован на продукт с упрощенной навигацией, большими кнопками, интуитивно понятными значками и логически сгруппированным контентом. Навигация, как правило, ясна и интуитивно понятна.

Функции поиска удобны для пользователя, часто включают автозаполнение, допуск опечаток и сами предлагают ключевые слова. Параметры фильтрации позволяют пользователям быстро уточнять результаты поиска по популярным критериям.

Все платформы имеют адаптивный кроссплатформенный дизайн. Это достигается с помощью гибких макетов сетки и плавной анимации. Кроме того, рассматриваемые платформы обеспечивают обратную связь с пользователем. Например, кнопки реагируют на действия пользователя, отображаются уведомления об успехах и ошибках, а продукты загружаются постепенно по мере прокрутки ленты пользователем.

Процесс оформления заказа оптимизирован, часто ограничен одним или двумя экранами с минимальным количеством полей ввода. Платформы поощряют быстрый вход с использованием номеров телефонов или учетных записей социальных сетей и предлагают варианты сохранения адресов и способов оплаты.

Некоторые платформы также реализуют уникальные интерфейсные решения. Например, Ozon предлагает интеллектуальную ленту персонализированных предложений и четко разделяет свою торговую площадку и продуктовый раздел. Wildberries известен своим сверхбыстрым поиском с мгновенным выводом результатов. Lamoda использует визуальные фильтры, такие как выбор цвета с помощью палитры, и ориентированный на моду макет, который выделяет визуальный контент.

Полученные в результате этого анализа результаты привели к нескольким важным выводам, которые послужили руководством для последующего тестирования и оптимизации платформы электронной коммерции. Пользователи ожидают быстрого доступа к информации и простого выполнения задач. Успешные платформы полагаются на устоявшиеся поведенческие модели. Адаптивность, скорость загрузки и минималистичный дизайн являются важнейшими характеристиками

современного интерфейса, в то время как персонализация и обратная связь значительно повышают уровень вовлеченности и доверия пользователей.

Полученные результаты легли в основу проектирования собственной E-commerce платформы и стали отправной точкой для определения критериев успешности при UX-тестировании и оценке результатов внедрения.

3.2 Проведение пользовательского тестирования

Пользовательское тестирование - это важный этап в процессе оценки качества интерфейса, нацеленный на выявление реального пользовательского опыта и проблем взаимодействия с платформой. В рамках данной дипломной работы было проведено тестирование разработанного интерфейса платформы электронной коммерции с целью определения его удобства, понятности и эффективности.

Целью пользовательского тестирования является получение обратной связи о работе интерфейса сайта: насколько удобно им пользоваться, легко ли выполнять базовые операции (добавлять товары в корзину, просматривать карточки товаров и редактировать профиль), и какие элементы требуют доработки или оптимизации.

Для тестирования была выбрана модераторская сессия с наблюдением, проводимая очно. Такой подход позволил фиксировать поведение пользователей, их комментарии и эмоции, а также оперативно задавать уточняющие вопросы. Также применялось тестирование по сценарию - пользователям предлагалось выполнить определенные задачи на сайте, после чего они заполняли короткую анкету с субъективной оценкой интерфейса.

В тестировании участвовали восемь человек разного возраста (от 18 до 45 лет), все пользователи имели опыт покупок в интернет-магазинах, но ранее не были знакомы с данной платформой. Половина респондентов пользовались тестируемым сайтом с мобильных устройств.

Участникам предлагалось выполнить серию типичных пользовательских сценариев: просмотр страницы товара, добавление товара в корзину, редактирование профиля и выход из системы. Все действия выполнялись без предварительных инструкций. Проводя сессии, фиксировали время выполнения задач, количество допущенных ошибок, затруднения и общий эмоциональный отклик участников.

Дополнительно после завершения сценариев участники заполняли короткую анкету, в которой оценивали интерфейс по ряду параметров, таких как удобство навигации, понятность структуры, визуальная привлекательность и общее впечатление от взаимодействия с платформой. Использовалась пятибалльная шкала, где 1 - крайне неудовлетворительно, а 5 - отлично. Результат тестирования представлен в таблице 2.

Таблица 2 – Результат тестирования

Участник	Удобство интерфейса	Навигация	Визуальное оформление	Общее впечатление
1	5	5	4	5
2	4	4	4	4
3	4	5	5	5
4	4	5	4	4
5	5	4	4	5
6	5	5	5	4
7	4	5	4	5
8	5	4	4	5

Результаты анкет показали высокий уровень удовлетворенности: в среднем пользователи оценили удобство интерфейса на 4.5 балла, навигацию - на 4.6, а визуальное оформление - на 4.4.

Большинство участников успешно справились с поставленными задачами без посторонней помощи. Отдельные сложности возникли у нескольких пользователей на мобильной версии: не сразу было понятно, как

добавить товар в избранное. Также отмечалось, что визуальной обратной связи при добавлении товара в корзину недостаточно - пользователи не всегда замечали, что их действие успешно выполнено.

Несмотря на эти мелкие замечания, в целом интерфейс был воспринят как современный, интуитивно понятный и достаточно отзывчивый. Пользователи отметили, что платформа напоминает им другие известные и привычные сервисы, что, в свою очередь, делает процесс адаптации к новому интерфейсу легким и быстрым.

Таким образом, пользовательское тестирование подтвердило правильность выбранных решений при проектировании интерфейса. Оно позволило выявить как сильные стороны реализации (простота, адаптивность, визуальная чистота), так и отдельные элементы, требующие уточнения и доработки. Все выявленные замечания были зафиксированы и частично устранены до завершения проекта. В целом, результаты тестирования свидетельствуют о высокой степени соответствия интерфейса ожиданиям и потребностям пользователей.

3.3 Оценка производительности и скорости загрузки интерфейса

Одним из ключевых факторов, влияющих на пользовательский опыт при работе с платформой электронной коммерции, является производительность веб-интерфейса. Под производительностью понимается скорость загрузки страниц, отклик на действия пользователя и общее ощущение работы веб-интерфейса.

Даже визуально привлекательный и функционально насыщенный интерфейс может вызывать негативные впечатления, если он работает медленно или нестабильно. Поэтому оценка производительности стала важным этапом тестирования и оптимизации разработанного интерфейса.

Платформа была реализована с использованием фреймворка Next.js, который изначально предлагает ряд решений, способствующих повышению

производительности: серверный рендеринг, автоматическую оптимизацию ресурсов и поддержку lazy-loading изображений. Все эти технологии были применены в проекте, однако для объективной оценки их эффективности было проведено измерение ключевых показателей с использованием инструментов анализа.

Основным инструментом для оценки скорости загрузки страниц и общего качества фронтенда стал Google Lighthouse - встроенный модуль Chrome DevTools, позволяющий проводить комплексную проверку веб-приложений по нескольким метрикам: производительность, доступность и удобство для мобильных устройств. Также использовались сервисы PageSpeed Insights и WebPageTest для более детального анализа загрузки конкретных ресурсов.

Тестирование производилось как на локальной версии проекта, так и на развернутом удаленном стенде, чтобы исключить искажения, связанные с особенностями локальной среды. Были проведены замеры для страницы товара и страницы редактирования профиля как на десктопе, так и на мобильных устройствах при стандартных параметрах сети.

Результаты показали, что страница товара загружалась в среднем за 1,3-1,6 секунды при первом посещении и менее чем за 1 секунду при повторной загрузке благодаря кэшированию и предварительной загрузке ресурсов. Скорость отображения первых визуальных элементов (FCP, от англ. First Contentful Paint - первое существенное отображение) составляла около 0,9 секунды, а интерактивность интерфейса (TTI , от англ. Time to Interactive – время до интерактивности) - около 1,5 секунды, что соответствует хорошим показателям по современным стандартам. Lighthouse выдавал оценку 90–95 баллов по показателю Performance.

Анализ результатов также позволил выявить области для оптимизации. Например, на отдельных страницах присутствовали изображения высокого разрешения без адаптации под экраны меньшего размера, что увеличивало общий объем загружаемых данных. Для решения этой проблемы был

внедрен компонент <Image> из Next.js, автоматически подстраивающий размеры и форматы изображений под устройство пользователя. Это позволило существенно сократить вес медиаконтента без потери качества и повысить скорость загрузки на мобильных устройствах.

Особое внимание было уделено работе с CSS. Благодаря использованию Tailwind CSS, в финальную сборку включались только реально используемые стили, а все лишнее удалялось на этапе сборки через механизм purge. Это значительно сократило размер итогового CSS-файла и положительно сказалось на времени загрузки и рендеринга.

Кроме того, проверялась реализация адаптивной верстки и отклик интерфейса на пользовательские действия. Интерфейс показывал стабильную и плавную работу при навигации, скроллинге, открытии всплывающих окон и переключении вкладок, как на десктопных, так и на мобильных устройствах. Анимации и переходы были реализованы с учетом оптимизации: использовались CSS-трансформации, а не тяжелые JavaScript-эффекты.

Таким образом, комплексная оценка производительности показала, что разработанный интерфейс соответствует современным требованиям по скорости загрузки, интерактивности и отзывчивости. Примененные архитектурные решения и инструменты оптимизации позволили достичь высоких показателей без ущерба для функциональности и визуального восприятия.

3.4 Оптимизация интерфейса на основе обратной связи

После первоначальной реализации и пользовательского тестирования E-commerce интерфейса на базе фреймворка Next.js важным этапом стало проведение анализа обратной связи и реализация улучшений, направленных на повышение удобства, производительности и функциональности. Такой итерационный подход позволяет не только устранять выявленные проблемы,

но и непрерывно адаптировать продукт к реальным потребностям пользователей.

Сбор обратной связи осуществлялся в нескольких форматах. Во-первых, был организован доступ к платформе для небольшой группы пользователей. Им было предложено пройти типичный сценарий использования: просмотр товаров, добавление в корзину, редактирование данных профиля. Полученные замечания были классифицированы по степени важности и срочности:

1. Удобство использования. Некоторые пользователи отмечали, что кнопка “Добавить в корзину” недостаточно выделялась на странице товара. В результате было изменено цветовое оформление и добавлен hover-эффект с анимацией. Страница профиля пользователя воспринималась перегруженной. Было принято решение упростить интерфейс, оставив только кнопки “Редактировать”, “Выйти из аккаунта” а также радиокнопку “Уведомлять об изменении статуса заказов по email”.

2. Функциональные предложения. По просьбам тестировщиков был добавлен индикатор загрузки при переходе между страницами, чтобы избежать ощущения “зависания”.

3. Производительность и отклик интерфейса. Отзывы пользователей на мобильных устройствах указывали на небольшие задержки при открытии модальных окон. После анализа выяснилось, что компоненты загружались синхронно с другими тяжелыми элементами. Решением стало оптимизация изображений на странице товара за счет использования формата WebP и установки размеров через компонент <Image /> из Next.js.

4. Эстетика и визуальные элементы. Изменены размеры иконок и межстрочные интервалы на мобильных устройствах - по итогам замечаний они воспринимались слишком мелкими. Изменены отступы и шрифты в некоторых секциях.

Каждое из указанных изменений проходило отдельное тестирование, как визуальное, так и функциональное. После нескольких циклов правок

интерфейс стал более логичным, визуально сбалансированным и удобным для конечного пользователя.

Также была внедрена система сбора анонимной статистики использования: какие разделы наиболее посещаемы, где происходят отказы, сколько времени тратится на определенные действия. Эти данные стали основой для планирования будущих итераций и дальнейшей оптимизации, например, упрощения пути от карточки товара до оформления заказа.

Таким образом, оптимизация интерфейса на основе обратной связи продемонстрировала высокую эффективность. Раннее вовлечение пользователей в процесс улучшения платформы позволило оперативно выявить слабые места, а итерационный подход к доработке сделал интерфейс более интуитивным, адаптивным и надежным. Этот процесс стал логичным завершением практической части дипломной работы и определил направления дальнейшего развития проекта.

3.5 Возможности дальнейшего развития проекта

Созданный в рамках дипломной работы интерфейс платформы электронной коммерции представляет собой полноценное и функциональное решение, удовлетворяющее современным требованиям к дизайну, адаптивности и производительности. Однако в условиях постоянного развития технологий, роста пользовательских ожиданий и изменений на рынке электронной коммерции, существует ряд направлений, по которым проект может развиваться в будущем. Эти перспективы охватывают как технические, так и пользовательские аспекты, а также открывают возможности для масштабирования и интеграции с внешними системами.

1. Расширение функциональности. Одним из очевидных шагов является добавление новых функций, которые повысят ценность платформы для пользователей и владельцев бизнеса. Среди таких функций можно выделить:

- Система рекомендаций на основе поведения пользователей, истории заказов и анализа интересов, реализуемая с помощью алгоритмов машинного обучения.

- Отзывы и рейтинги к товарам, что повысит уровень доверия к платформе и поможет другим пользователям принимать решения о покупке.

- Продвинутый поиск с учетом синонимов, фильтрации по нескольким критериям, поддержки ошибок в наборе текста.

- Сравнение товаров, позволяющее пользователю наглядно сопоставлять характеристики и выбирать оптимальный вариант.

- Список желаний, куда можно добавлять понравившиеся товары.

2. Административная панель. На текущем этапе интерфейс ориентирован преимущественно на конечного пользователя. Однако для полноценного управления платформой необходимо реализовать административную панель, которая позволит управлять товарами (добавление, редактирование, удаление, сортировка), обрабатывать заказы, отслеживать статусы доставок, управлять пользователями (права доступа), анализировать статистику продаж, статистику посещаемости, коэффициенты конверсии и другие метрики.

Разработка административного интерфейса также может опираться на фреймворки Next.js и Tailwind CSS, обеспечивая единство архитектуры приложения.

3. Поддержка мультиязычности. Для выхода на международный рынок потребуется реализация поддержки нескольких языков интерфейса. Это возможно с помощью возможностей библиотеки i18n в Next.js [20]. Интернационализация включает:

- Перевод текстов и интерфейсных элементов.

- Локализацию форматов дат, валют, чисел.

- Адаптацию дизайна под особенности восприятия в разных регионах.

4. Интеграция с внешними сервисами. Следующий логичный шаг - расширение взаимодействия с внешними платформами:

- Платежные системы (Сбербанк, ЮKassa и др.).

- Службы доставки (CDEK, Boxberry, Почта России) с возможностью отслеживания посылок.

- CRM-системы для управления взаимоотношениями с клиентами и автоматизацией маркетинга.

- Системы аналитики (Google Analytics, Yandex Metrika,) для углубленного анализа поведения пользователей.

5. Оптимизация производительности и SEO. С учетом роста количества пользователей и объема данных необходимо будет работать над улучшением времени загрузки страниц, особенно на мобильных устройствах. Так же необходимо осуществить оптимизацию изображений и кэширование, а также расширение семантики HTML для улучшения результатов в поисковых системах.

6. Переход к прогрессивному веб-приложению (PWA, от англ. Progressive Web App – прогрессивное веб-приложение). Добавление возможностей офлайн-доступа, push-уведомлений и установки сайта как приложения на устройства пользователя сделает платформу ближе к нативному приложению, увеличив лояльность и вовлеченность клиентов. Для этого можно использовать встроенные возможности Next.js и библиотеки для PWA [21].

7. Развитие архитектуры. В перспективе роста проекта можно перейти на микроархитектуру, что упростит масштабирование команды и отдельных компонентов интерфейса. Также возможна реализация более сложной архитектуры взаимодействия с сервером, включая подписки на обновления и realtime-функции (например, уведомления о наличии товара).

Таким образом, разработанный интерфейс является лишь первым этапом масштабного проекта. Благодаря использованию современных технологий и модульной архитектуры он уже сейчас готов к

масштабированию и может быть основой для построения мощной и гибкой платформы электронной коммерции. Реализация предложенных направлений развития позволит не только повысить технический уровень продукта, но и существенно улучшить пользовательский опыт, адаптируя платформу под нужды растущего бизнеса и аудитории.

Выводы к разделу 3

На этапе тестирования, оценки и оптимизации пользовательского интерфейса были проведены мероприятия, направленные на повышение удобства использования, скорости работы и общей эффективности взаимодействия пользователя с платформой электронной коммерции.

В результате анализа существующих решений и UX-практик конкурентов были выявлены типовые подходы к организации пользовательского интерфейса, наиболее часто используемые паттерны и элементы взаимодействия, соответствующие ожиданиям современных пользователей. Эти данные стали основой для выстраивания логичной, интуитивной и визуально понятной структуры платформы.

Пользовательское тестирование позволило на практике оценить, насколько созданный интерфейс удобен, понятен и функционален. Были выявлены неочевидные проблемы в навигации, визуальном восприятии элементов и логике некоторых взаимодействий. В ходе итерационного подхода эти проблемы были устранены, что существенно повысило общее качество интерфейса. Благодаря обратной связи от реальных пользователей удалось учесть их потребности и адаптировать функциональность под реальные сценарии использования.

Отдельное внимание было уделено оценке производительности интерфейса. Были измерены ключевые метрики: скорость загрузки страниц, время отклика интерфейсных элементов, объем передаваемых данных.

Оптимизация изображений позволила значительно улучшить эти показатели, особенно в мобильных условиях и при слабом интернет-соединении.

Проект также был адаптирован под различные устройства и браузеры. Проведено кроссбраузерное тестирование и настройка адаптивной верстки, что обеспечило корректную работу интерфейса как на десктопах, так и на мобильных устройствах. Это особенно важно для E-commerce платформ, так как значительная часть пользователей делает заказы через мобильные устройства.

На завершающем этапе была реализована итеративная оптимизация на основе собранной обратной связи. Мелкие улучшения, такие как изменения в иконках, отступах, поведении модальных окон, а также добавление вспомогательных элементов (например, индикатор загрузки), привели к более целостному и проработанному пользовательскому опыту.

Таким образом, тестирование и последующая оптимизация стали неотъемлемой частью процесса разработки интерфейса. Они позволили не только выявить и устранить недочеты, но и значительно улучшить взаимодействие пользователя с платформой. Проведенная работа доказала важность итерационного подхода и тесной связи с конечными пользователями при создании современных веб-приложений. Полученные результаты создают основу для дальнейшего масштабирования проекта и внедрения новых функций.

Заключение

В ходе выполнения дипломной работы была решена комплексная задача по созданию и оптимизации пользовательского интерфейса платформы электронной коммерции на базе современного стека технологий. Основной акцент был сделан на удобство, производительность и адаптивность интерфейса, что соответствует текущим требованиям рынка электронной коммерции и ожиданиям пользователей.

На этапе анализа (Раздел 1) был рассмотрен современный подход к проектированию пользовательских интерфейсов в сфере создания E-commerce платформ. Выделены основные особенности и задачи таких интерфейсов, проанализированы современные технологии веб-разработки, влияние пользовательского опыта на эффективность онлайн-продаж, а также важность адаптивной и кроссплатформенной верстки. Отдельное внимание было уделено работе с прототипами интерфейса в Figma, на основе которого и велась дальнейшая реализация. Постановка задачи позволила четко сформулировать цели и критерии эффективности интерфейса.

В рамках проектной части (Раздел 2) были обоснованы архитектурные и технологические решения, подробно изучены особенности фреймворка Next.js, его структура и возможности маршрутизации. Были реализованы основные интерфейсные компоненты, разработан функционал страницы товара и страницы профиля, реализовано взаимодействие с серверной частью через API. Весь код был написан с использованием компонентного подхода и типизирован на TypeScript, что обеспечило надежность и расширяемость проекта. Для стилизации применен Tailwind CSS, благодаря чему удалось обеспечить визуальную целостность и адаптивность интерфейса.

На завершающем этапе (Раздел 3) был проведен комплекс тестов, включающий UX-анализ решений конкурентов, пользовательское тестирование, оценку производительности, адаптацию интерфейса под различные устройства и браузеры. Оптимизация проекта происходила

итерационно, на основе обратной связи от реальных пользователей. Особое внимание было уделено быстродействию, что является критически важным параметром для платформ электронной коммерции. Кроме того, были определены перспективы дальнейшего развития проекта, включая расширение функциональности, добавление системы рекомендаций, мультиязычности, административной панели и интеграцией с внешними сервисами.

Таким образом, все цели, поставленные в начале дипломной работы, были достигнуты. Разработанный интерфейс представляет собой современное, удобное и масштабируемое решение, пригодное для практического использования в коммерческих проектах. Работа продемонстрировала важность целостного проектирования интерфейса и тесной связи с конечными пользователями на всех этапах разработки.

Полученные результаты подтверждают, что применение фреймворка Next.js совместно с TypeScript и Tailwind CSS позволяет эффективно решать задачи создания высококачественных пользовательских интерфейсов для E-commerce платформ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Ассоциация компаний интернет-торговли (АКИТ). Сводные аналитические данные [Электронный ресурс]. URL: https://akit.ru/analytics/analyt-data (дата обращения 17.04.2025).

2 INFOLine. Прогноз на 2025 год: онлайн-торговля в России вырастет до 14,9 трлн рублей на фоне консолидации рынка [Электронный ресурс]. URL: https://inforetail.ru/ (дата обращения 25.03.2025).

3 What is E-Commerce? [Электронный ресурс]. URL: https://www.techopedia.com/definition/ecommerce (дата обращения 10.04.2025).

4 Gualtieri M. Leaving User Experience To Chance Hurts Companies [Электронный ресурс]. URL: https://www.forrester.com/blogs/09-10-15-leaving_user_experience_to_chance_hurts_companies/ (дата обращения 10.04.2025).

5 Ecommerce User Experience – 5th Edition. Nielsen Norman Group [Электронный ресурс]. URL: https://www.nngroup.com/reports/ecommerce-user-experience/ (дата обращения 15.04.2025).

6 Web Content Accessibility Guidelines (WCAG) [Электронный ресурс]. URL: https://www.w3.org/WAI/standards-guidelines/wcag/ (дата обращения 15.04.2025).

7 React documentation [Электронный ресурс]. URL: https://react.dev/learn (дата обращения 25.03.2025).

8 Next.js documentation. Vercel, 2023 [Электронный ресурс]. URL: https://nextjs.org/docs (дата обращения 27.03.2025).

9 Remix Docs. Remix, 2023 [Электронный ресурс]. URL: https://remix.run/docs (дата обращения 05.04.2025).

10 Gatsby Docs. Gatsby, 2023 [Электронный ресурс]. URL: https://www.gatsbyjs.com/docs/ (дата обращения 02.04.2025).

11 Astro Docs. Astro, 2023 [Электронный ресурс]. URL: https://docs.astro.build/ (дата обращения 04.04.2025).

12 Copes F. The Svelte Handbook. Harris, 2022 [Электронный ресурс]. URL: https://flaviocopes.com/page/svelte-handbook/ (дата обращения 01.05.2025).

13 Nuxt Docs. Nuxt Labs, 2023 [Электронный ресурс]. URL: https://nuxt.com/docs (дата обращения 04.04.2025).

14 Tailwind CSS Documentation [Электронный ресурс]. URL: https://tailwindcss.com/docs (дата обращения 27.03.2025).

15 ISO 9241-210:2019. Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems [Электронный ресурс]. URL: https://www.iso.org/standard/77520.html (дата обращения 10.04.2025).

16 Forrester Research. The Business Value of User Experience [Электронный ресурс]. URL: https://go.forrester.com (дата обращения 10.04.2025).

17 Статистика онлайн-покупок: аналитика и тенденции [Электронный ресурс] // OptinMonster. URL: https://optinmonster.com/online-shopping-statistics/ (дата обращения 20.03.2025).

18 Рекомендации по мобильной индексации [Электронный ресурс] // Google Developers. URL: https://developers.google.com/search/docs/crawling-indexing/mobile/mobile-sites-mobile-first-indexing (дата обращения 20.05.2025).

19 Figma Documentation [Электронный ресурс]. URL: https://figma.com (дата обращения 24.03.2025).

20 Next.js Internationalization Guide [Электронный ресурс]. URL: https://nextjs.org/docs/pages/guides/internationalization (дата обращения 15.04.2025).

21 Next.js Progressive Web Apps Guide [Электронный ресурс]. URL: https://nextjs.org/docs/app/guides/progressive-web-apps (дата обращения 12.04.2025).

22 The History Of Ecommerce: How Did It All Begin? [Электронный ресурс]. URL: https://blog.miva.com/the-history-of-ecommerce-how-did-it-all-begin (дата обращения 17.03.2025).

23 Ecommerce sales growth in 2024 [Электронный ресурс]. URL: https://www.oberlo.com/statistics/global-ecommerce-sales-growth (дата обращения 28.03.2025).

24 E-commerce market size in Russia from 2011 to 2027 [Электронный ресурс]. URL: https://www.statista.com/statistics/1016094/russia-e-commerce-market-value/ (дата обращения 14.03.2025).

25 Sees the Upwards Trend of Russia's E-commerce Market with Considerable Highlights [Электронный ресурс]. URL: https://equalocean.com/analysis/2023070519847 (дата обращения 12.04.2025).

26 Ecommerce User Experience – 5th Edition. Nielsen Norman Group [Электронный ресурс]. URL: https://www.nngroup.com/reports/ecommerce-user-experience/ (дата обращения 21.03.2025).

27 Александр Сидоров. Wildberries: «AI помогает маркетплейсам создавать возможности» [Электронный ресурс]. URL: https://www.forbes.ru/brandvoice/526438-aleksandr-sidorov-wildberries-ai-pomogaet-marketplejsam-sozdavat-vozmoznosti (дата обращения 17.03.2025).

28 Дон Норман. «The Design of Everyday Things». – М.: “МАНН, ИВАНОВ И ФЕРБЕР”, 2018 – 384 с.

29 The Next.js Handbook: A Complete Resource for Developers. – Ares Lnc, 2023 – 108 с.

30 Ultimate Tailwind CSS Handbook: Build sleek and modern websites with immersive UIs using Tailwind CSS. – Orange Education Pvt Ltd, 2023 – 293 с.

Приложение 1

Код компонента Product для страницы товара приложения электронной коммерции

```
import HomeContainer from "@/components/HomeContainer/HomeContainer";
import ProductPage from "@/components/ProductPage/ProductPage";
import {  GetServerSideProps } from "next";
import { useProductById } from "../../shared/hooks/queries/useProductById";
import LoadingIcon from "../../components/LoadingIcon/LoadingIcon";
import { getFavoritesInfo } from "@/shared/api/products";
import { useQuery } from "@tanstack/react-query";
import { useEffect, useState } from "react";
import { ProductInfo } from "@/shared/types";
import { useUserStore } from "@/shared/store/auth";

type Product = {
  id: number;
  title: string;
  price: number;
  description: string;
  availability: boolean;
};

type Props = {
  productId: number;
};

export default function Product(props: Props) {
  const { productId } = props;
  const {product} = useProductById(productId);
  const { isAuthenticated } = useUserStore();
  const [favoritesItems, setFavoritesItems] = useState<Product[]>([]);

  const { data } = useQuery({
    queryKey: ["favoritesInfo"],
    queryFn: getFavoritesInfo,
  });
  const favorites = data?.likedItems ?? [];

  useEffect(() => {
      const favoritesItemsFormatted: Product[] = favorites.map(
        (fav: ProductInfo) => fav.item,
```

```
      );
      setFavoritesItems(favoritesItemsFormatted);
    }, [isAuthenticated]);

  return (
    <HomeContainer>
      {product ? <ProductPage
        product={product.item}
        characteristics={product.characteristics}
        roundRating={product.averageRating}
        quantityRatings={product.postsCount}
        photos={product.photos}
        favorites={favoritesItems}
      /> : <LoadingIcon />}
    </HomeContainer>
  );
}

export const getServerSideProps: GetServerSideProps = async (context) => {
  const { id } = context.params!;
  const productId = parseInt(id as string, 10);
  if (isNaN(productId)) {
    return {
      redirect: {
        destination: "/",
        permanent: false,
      },
    };
  }

  return {
    props: {
      productId,
    },
  };
};
```

Код компонента ProductPage для страницы товара приложения электронной коммерции

```
import Breadcrumbs from "../Breadcrumbs/Breadcrumbs";
import Image from 'next/image';
import { useState } from "react";
import { getRatingsWord } from "@/shared/utils/frontend/cartHelpers";
import { HeartIcon as HeartIconBlack } from "@heroicons/react/24/solid";
import { handleToggleFavorite } from "@/shared/utils/frontend/fetch";
import { addProductInBacket } from "@/shared/api/basket";
import { useBasket } from "@/shared/hooks/queries/useBasket";
import ReplaceQuantity from "../MiniCard/ReplaceQuantity";

type Product = {
  id: number;
  title: string;
  price: number;
  description: string;
  availability: boolean;
};

type Characteristic = {
  id: number;
  itemId: number;
  color: string;
  frameMatherials: string;
  linzeMatherials: string;
  linzeTypes: string;
  linzeUVDefences: string;
  linzeEffects: string;
};

type Photo = {
  id: number;
  itemId: number;
  photoLink: string;
  isMainPhoto: boolean;
};

type ProductPageProps = {
  product: Product;
  characteristics: Characteristic[];
```

```
    roundRating: number;
    quantityRatings: number;
    photos: Photo[];
    favorites: Product[] | [];
  };

  export default function ProductPage({
    product,
    characteristics,
    roundRating,
    quantityRatings,
    photos,
    favorites,
  }: ProductPageProps) {

     const defaultMainPhoto = photos.find(p => p.isMainPhoto)?.photoLink ||
"/images/product_for_dev.png";
                const       [mainPhotoLink,       setMainPhotoLink]       =
useState<string>(defaultMainPhoto);
              const      nonMainPhotos      =      photos.filter(photo      =>
!photo.isMainPhoto).slice(0, 4);
     const availabilityProduct = product.availability ? "Есть в наличии" :
"Нет в наличии";
    const ratingsWord = getRatingsWord(quantityRatings);
    const basketQuery = useBasket();
    const itemId = product.id || 0;
    const inBasket = basketQuery.basket?.items
     ? basketQuery.basket.items.some((basketItem) => basketItem.itemId ===
itemId)
    : false;

    const characteristicsList = characteristics[0]
    ? {
        "Цвет": characteristics[0].color,
        "Материал оправы": characteristics[0].frameMatherials,
        "Линзы": characteristics[0].linzeEffects,
        "Материал линз": characteristics[0].linzeMatherials,
        "Тип линз": characteristics[0].linzeTypes,
        "Наличие УФ фильтра": characteristics[0].linzeUVDefences,
      }
    : {};
```

```
    const liked = favorites
    ? favorites.some((favoriteItem) => favoriteItem.id === itemId)
    : false;
    const [isLiked, setIsLiked] = useState(liked);
    const [quantity, setQuantity] = useState(1);
    console.log(' liked.id ',  liked);

    function handleLikeClick() {
      setIsLiked((prev) => !prev);

      handleToggleFavorite(itemId)
        .then(() => {

        })
        .catch((err) => {
          setIsLiked((prev) => !prev);
          console.log(`Ошибка: ${err}`);
        });
    }

    function addToCart() {
        setQuantity(1);
        addProductInBacket({ itemId, quantity: 1 })
          .then((response) => {
            console.log("Товар добавлен в корзину:", response.message);
          })
          .catch((error) => {
                console.error("Ошибка при добавлении товара в корзину:",
error.message);
          });
      }

    return (
      <div className="mx-auto max-w-[980px]">
        <Breadcrumbs className="hidden md:block md:pb-5 md:pt-0"/>
         <div className="relative flex flex-wrap justify-center gap-4 mb-4
md:mb-8 md:rounded-xl md:bg-white md:p-6 md:shadow-custom">

            <div className="px-5 md:px-0 gap-5 flex flex-col items-center
md:items-start md:flex-row md:justify-center">

            <div>
```

```
            <div className="min-w-[335px] min-h-[288px] bg-white rounded-
lg overflow-hidden flex items-center justify-center mt-5 md:mt-0 mb-3 md:mb-
0">
              <Image
                src={mainPhotoLink}
                alt="Основная фотография очков"
                width={456}
                height={460}
                className="w-full h-full object-cover"
              />
            </div>

            <div className="hidden md:flex justify-center gap-2 mt-4">
              {nonMainPhotos.map((photo) => (
                <div
                  key={photo.id}
                      className="w-[97px] h-[96px] bg-gray-200 rounded-lg
hover:cursor-pointer"
                  onClick={() => setMainPhotoLink(photo.photoLink)}
                >
                  <Image
                    src={photo.photoLink}
                    alt="Другие фотографии очков"
                    width={97}
                    height={96}
                    className="w-full h-full object-cover"
                  />
                </div>
              ))}
            </div>
          </div>

          <main className="w-full max-w-[456px]">
            <div className="flex justify-between">
              <h1 className="font-bold text-[24px] md:text-[30px] leading-
8 md:leading-9 text-[#1F2937]">{product.title}</h1>

              <div className="flex flex-col items-center justify-center
gap-1">
              <div className="flex items-center gap-2">
                <Image
                       className="w-4 h-4 md:w-8 md:h-8 text-[#2563EB]
```

```
object-center object-contain"
                      src="/images/Star.svg"
                      alt="Рейтинг"
                      width={32}
                      height={32}
                    />
                      <span className="font-bold text-[24px] md:text-[30px]
leading-8 md:leading-9 text-[#1F2937]">{roundRating}</span>
                  </div>
                </div>

              </div>
              <div className="flex justify-end">
                    <span  className="mt-0  md:mt-1  font-normal  text-[14px]
leading-5 text-[#6B7280]">{ratingsWord}</span>
              </div>

               <div className="text-[#10B981] font-bold text-[24px] md:text-
[30px] leading-8 md:leading-9">{product.price} ₽</div>

              <div className="hidden md:flex mt-4 justify-between gap-4">

                <div className="flex gap-1">

                  {inBasket ? (
                    <div className="flex flex-grow justify-end gap-2">
                      <ReplaceQuantity
                        id={itemId}
                        quantity={quantity}
                        onQuantityChange={(newQty) => setQuantity(newQty)}
                      />
                    </div>
                  ) : (
                    <button className="bg-[#1E40AF] w-[180px] h-[40px] text-
white rounded-[6px] flex items-center justify-center">
                      <Image
                        src="/images/button_bag.svg"
                        alt="Корзина"
                        width={24}
                        height={24}
                        className="w-6 h-6"
                        onClick={addToCart}
```

```
                    />
                  </button>
                )}

                  <button className="flex w-[40px] h-[40px] items-center
justify-center">
                {isLiked ? (
                  <HeartIconBlack width={24} height={24} />
                ) : (
                  <Image
                    className="w-5 h-[18px] text-[#1E40AF] object-center
object-contain"
                    src="/images/Heart.svg"
                    alt="Лайк"
                    width={20}
                    height={18}
                    onClick={handleLikeClick}
                  />
                )}
              </button>
            </div>

                  <p className="font-normal text-[16px] leading-6 text-
[#6B7280] self-end">{availabilityProduct}</p>
          </div>

              <p className="mt-3 md:mt-4 font-bold text-[14px] md:text-
[16px] leading-5 md:leading-6 text-[#1F2937]">Описание</p>
            <p className="mt-1 md:mt-4 font-normal text-[12px] md:text-
[14px] leading-4 md:leading-5 text-[#6B7280]">
              {product.description}
          </p>
              <h2 className="mt-3 md:mt-4 font-bold text-[14px] md:text-
[16px] leading-5 md:leading-6 text-[#1F2937]">О товаре</h2>

          <div className="mt-[6px] md:mt-[14px]">
                {Object.entries(characteristicsList).map(([key, value],
index, array) => (
              <div
                key={key}
                className={`flex justify-between ${
                    index === array.length - 1 ? 'pt-[6px] md:pt-2' :
```

```
index === 0 ? 'border-b pb-[6px] md:pb-2': 'border-b py-[6px] md:py-2'
                    }`}
                  >
                    <span className="font-normal text-[12px] leading-4 text-
[#6B7280]">{key}</span>
                     <span className="font-normal text-[14px] md:text-[16px]
leading-5 md:leading-6 text-[#1F2937]">{value}</span>
                  </div>
                ))}
              </div>
              <div className="block md:hidden mt-4 gap-4">

                <div className="flex justify-between gap-0">

                  {inBasket ? (
                    <div className="flex flex-grow justify-end gap-2">
                      <ReplaceQuantity
                        id={itemId}
                        quantity={quantity}
                        onQuantityChange={(newQty) => setQuantity(newQty)}
                      />
                    </div>
                  ) : (
                      <button className="bg-[#1E40AF] flex-1 h-[40px] text-
white rounded-[6px] flex items-center justify-center">
                    {product.availability ?
                    <Image
                      src="/images/button_bag.svg"
                      alt="Корзина"
                      width={24}
                      height={24}
                      className="w-6 h-6"
                      onClick={addToCart}
                    />
                    : "Нет в наличии"
                    }
                  </button>
                  )}
                     <button className="flex w-[40px] h-[40px] items-center
justify-center">
                    {isLiked ? (
                      <HeartIconBlack width={24} height={24} />
```

```
                    ) : (
                      <Image
                        className="w-5 h-[18px] text-[#1E40AF] object-center
object-contain"
                        src="/images/Heart.svg"
                        alt="Лайк"
                        width={20}
                        height={18}
                        onClick={handleLikeClick}
                      />
                    )}
                  </button>
                </div>
              </div>
            </main>
          </div>
        </div>
      </div>
    );
  }
```

Код компонента Profile для страницы редактирования профиля приложения электронной коммерции

```
import { useState, useEffect } from "react";
import HomeContainer from "@/components/HomeContainer/HomeContainer";
import ProfileBackground from "@/components/ProfileComponents/ProfileBackground";
import Sidebar from "@/components/Sidebar/Sidebar";
import ProfileSection from "@/components/ProfileSection/ProfileSection";
//import CustomSelect from "@/components/CustomSelect/CustomSelect";
import { useUserStore } from "@/shared/store/auth";
import { useMutation } from '@tanstack/react-query';
import { updateUser } from "@/shared/api/user";
import { useRouter } from 'next/router';
import { logOut } from "@/shared/api/auth";
import ProtectedRoute from "../../components/ProtectedRoute/ProtectedRoute";
import { useProtectedRoute } from "../../shared/hooks/useProtectedRoute";

export default function Profile() {

  const name = useUserStore((state) => state.name);
  const surname = useUserStore((state) => state.surname);
  const email = useUserStore((state) => state.email);
//  const isAuthenticated = useUserStore((state) => state.isAuthenticated);

  const [isEditing, setIsEditing] = useState(false);
  const [profileData, setProfileData] = useState({ name, surname, email });
  //const [isSidebarOpen, setIsSidebarOpen] = useState(false);

  const router = useRouter();

  const handleLogout = async () => {
    try {
      await logOut();
    } catch (error) {
      console.error("Ошибка при выходе:", error);
    } finally {
      useUserStore.getState().removeUserData();
      useUserStore.getState().setIsAuthenticated(false);
      router.push("/");
    }
```

```
};

const mutation = useMutation({
  mutationFn: updateUser,
  onSuccess: (_, variables) => {
    useUserStore.getState().setUserData({
      name: variables.name ?? "",
      surname: variables.surname ?? "",
      email: variables.email ?? "",
      avatar: "",
    });
    setIsEditing(false);
    console.log('Профиль успешно обновлён');
  },
  onError: (error) => {
    console.error('Ошибка при обновлении профиля:', error);
  },
});

const handleSave = (updatedData: {
  name: string;
  surname: string;
  email: string
}) => {
  setProfileData(updatedData);
  mutation.mutate(updatedData);
  setIsEditing(false);
};

useEffect(() => {
  setProfileData({ name, surname, email });
}, [name, surname, email]);

return (
  <HomeContainer>
    <ProtectedRoute protection={useProtectedRoute}>
      <ProfileBackground imageUrl="/images/sport_meditation.svg">
        <div className="flex h-[90vh] p-5 md:pt-10">
          <Sidebar />

          <main className="flex-1 pl-1 md:pl-5">
               <h1 className="mb-4 block text-[24px] font-bold leading-8
```

```
text-[#1F2937] md:hidden">
                Мой профиль
              </h1>

              <ProfileSection
                {...profileData}
                isEditing={isEditing}
                onEdit={() => setIsEditing(true)}
                onCancel={() => setIsEditing(false)}
                onSave={handleSave}
              />

              {!isEditing && (
                <div className="mt-4">
                        {/*<label  className="block  text-[12px]  font-normal
leading-4 text-[#6B7280] md:text-[14px] md:leading-5">
                    Язык:
                  </label>

                  <CustomSelect />*/}

                  <div className="mt-4 flex items-center space-x-2">
                    <input
                      type="checkbox"
                      id="emailNotify"
                        className="h-[16px] w-[16px] cursor-pointer rounded-
full border border-[#9CA3AF] bg-[#F9FAFB]"
                    />
                    <label
                      htmlFor="emailNotify"
                         className="text-[12px] font-normal leading-4 text-
[#1F2937] md:text-[14px] md:leading-5"
                    >
                      Уведомлять об изменении статуса заказов по email
                    </label>
                  </div>

                  {/*<div className="mt-4 flex items-center">
                    <input
                      type="checkbox"
                      id="darkModeToggle"
                      className="peer sr-only"
```

```
                    />

                    <label
                      htmlFor="darkModeToggle"
                        className="flex h-5 w-10 cursor-pointer items-center rounded-full bg-[#9CA3AF] p-[2px] transition-colors peer-checked:bg-gray-600"
                    >
                        <span className="h-4 w-4 transform rounded-full bg-white shadow-md transition-transform duration-300 peer-checked:translate-x-5"></span>
                    </label>

                    <label
                      htmlFor="darkModeToggle"
                          className="ml-2 text-[12px] font-normal leading-4 text-[#1F2937] md:text-[14px] md:leading-5"
                    >
                      Ночная тема
                    </label>
                  </div>*/}

                    <button
                      onClick={handleLogout}
                       className="w-full max-w-[720px] md:max-w-[580px] mt-4 px-8 py-[7px] bg-[#FFFFFF] text-[#1E40AF] border border-[#1E40AF] rounded-md font-[700] text-[14px] leading-5 md:text-[16px] md:leading-6 hover:bg-[#1E40AF] hover:text-white transition">
                        Выйти из аккаунта
                    </button>

                </div>
              )}
            </main>
          </div>
        </ProfileBackground>
      </ProtectedRoute>
    </HomeContainer>
  );
}
```

Код компонента ProfileSection для страницы редактирования профиля приложения электронной коммерции

```
import { useState, useEffect  } from "react";
import Image from "next/image";
import Avatar from "../../../public/images/initial_avatar.png";
import CameraIcon from "../../../public/icons/camera_icon.svg";

interface ProfileSectionProps {
    name: string;
    surname: string;
    email: string;
    isEditing: boolean;
    onEdit: () => void;
     onSave: (data: { name: string; surname: string; email: string }) =>
void;
    onCancel: () => void;
}

export default function ProfileSection({
    name,
    surname,
    email,
    isEditing = false,
    onSave,
    onCancel,
    onEdit,
  }: ProfileSectionProps) {
    const [formData, setFormData] = useState({ name, surname, email });

    useEffect(() => {
        setFormData({ name, surname, email });
    }, [name, surname, email]);

    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
         setFormData((prev) => ({ ...prev, [e.target.name]: e.target.value
}));
      };

    return (
        <>
```

```
            {isEditing ? (

                    <div className="bg-white pt-6 pb-6 pl-4 pr-4 rounded-xl
shadow-lg max-w-[720px] md:max-w-[580px]">
                     <div className="flex justify-center md:block">
                         <div className="relative w-[100px] h-20">
                                <Image className="rounded-full" src={Avatar}
width={80} height={80} alt="Avatar" />
                               <div className="absolute bottom-0 right-0 bg-
[#1E40AF] p-2 rounded-full cursor-pointer">
                                        <Image  src={CameraIcon}  width={24}
height={24} alt="Edit avatar" />
                            </div>
                         </div>
                     </div>


                     <div className="mt-4">
                         <div className="flex flex-col md:flex-row gap-4">

                            <div className="w-full md:w-1/2">
                                  <label className="block font-normal text-
[12px] leading-4 md:text-[14px] md:leading-5 text-[#6B7280]">Имя:</label>
                                <input
                                    type="text"
                                    name="name"
                                    value={formData.name}
                                    onChange={handleChange}
                                       className="w-full px-3 py-[7px] font-
normal  text-[14px]  leading-5  md:text-[16px]  md:leading-6  border  border-
[#9CA3AF]  bg-[#F3F4F6]  rounded-[4px]  text-gray-800  focus:ring  focus:ring-
blue-300"
                                />
                            </div>

                            <div className="w-full md:w-1/2">
                                  <label className="block font-normal text-
[12px]       leading-4       md:text-[14px]       md:leading-5       text-
[#6B7280]">Фамилия:</label>
                                <input
                                    type="text"
                                    name="surname"
```

```
                                value={formData.surname}
                                onChange={handleChange}
                                   className="w-full px-3 py-[7px] font-
normal text-[14px] leading-5 md:text-[16px] md:leading-6 border border-
[#9CA3AF] bg-[#F3F4F6] rounded-[4px] text-gray-800 focus:ring focus:ring-
blue-300"
                                />
                            </div>
                        </div>

                        <div className="mt-4 md:mt-2 w-full md:max-w-[266px]
md:w-[48.5%]">
                             <label className="block font-normal text-[12px]
leading-4 md:text-[14px] md:leading-5 text-[#6B7280]">Email:</label>
                            <input
                                type="email"
                                name="email"
                                value={formData.email}
                                onChange={handleChange}
                                 className="w-full px-3 py-[7px] font-normal
text-[14px] leading-5 md:text-[16px] md:leading-6 border border-[#9CA3AF]
bg-[#F3F4F6] rounded-[4px] text-gray-800 focus:ring focus:ring-blue-300"
                            />
                        </div>

                         <div className="flex flex-col md:flex-row justify-
start gap-4 md:gap-2 mt-10 md:mt-6">
                            <button
                                    className="px-4 py-[7px] border border-
[#1E40AF] text-[#1E40AF] rounded-md font-[700] text-[14px] leading-5
md:text-[16px] md:leading-6 hover:bg-blue-50 transition"
                                onClick={onCancel}
                            >
                                Отменить
                            </button>
                            <button
                                 className="px-8 py-[7px] bg-[#1E40AF] text-
white rounded-md font-[700] text-[14px] leading-5 md:text-[16px] md:leading-
6 hover:bg-blue-700 transition"
                                onClick={() => onSave(formData)}
                            >
                                Сохранить
```

```
                            </button>
                        </div>
                    </div>

                </div>

            ) : (

                    <div className="bg-white p-4 rounded-xl shadow-lg flex md:flex-row flex-col md:items-center items-start justify-between max-w-[720px] md:max-w-[580px]">
                     <div className="flex items-center space-x-4 mb-6 md:mb-0">
                                <Image className="rounded-full" src={Avatar} width={80} height={80} alt="Avatar" />
                            <div className="md:text-lg font-medium leading-6 text-[#1F2937] space-y-2 text-sm">
                        <h2>{`${name} ${surname}`}</h2>
                            <p>{email}</p>
                        </div>
                    </div>
                    <button
                        className="flex justify-center w-full md:w-auto min-w-[303px] md:min-w-[157px] py-[7px] bg-white text-[#1E40AF] rounded-[6px] border border-[#1E40AF] font-bold text-sm md:text-[16px] md:leading-6 transition-colors duration-200 hover:bg-[#1E40AF] hover:text-white"
                        onClick={onEdit}
                        >Редактировать
                    </button>
                </div>

            )}
        </>
    );
  };
```