МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

Направление подготовки
09.03.01 Информатика и вычислительная техника

Направленность (профиль)
«Технологии разработки программного обеспечения»

Выпускная квалификационная работа

Разработка SPA (одностраничного приложения) для онлайн-магазина с применением CSR и React

Обучающегося 4 курса
очной формы обучения
Каргаполова Дениса Андреевича

Руководитель выпускной квалификационной работы:
старший преподаватель кафедры информационных технологий и электронного обучения
Государев Илья Борисович

Санкт-Петербург
2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время рынок веб-разработки является очень требовательным к качеству, удобству и скорости работы продуктов. Это особенно актуально для онлайн-магазинов – сервисы, от которых пользователи ожидают не просто широкий функционал, но и максимально понятный интерфейс приложения. Самым наиболее эффективным подходом к созданию приложений такого типа являются одностраничные приложения (или же SPA, Single Page Application), которые работают по принципу клиентского рендеринга (CSR, Client-Side Rendering). Данный подход предлагает быструю загрузку страниц, плавную навигацию без полной перезагрузки, а также высокий уровень интерактивности.

SPA отличается от традиционных многостраничных решений тем, что единожды загружает основное содержимое сайта, после чего все взаимодействие происходит на стороне клиента. Данный способ значительно ускоряет работу интерфейса и уменьшает нагрузку на сервер. Технология CSR в этом контексте играет важную роль, обеспечивая динамическое обновление данных и отображение контента без перезагрузки страницы. Такой подход выгоден для онлайн-магазинов, где необходима высокая скорость отклика и постоянное обновление данных.

Наиболее популярной библиотекой для реализации SPA с CSR является React – JavaScript-библиотека. React даёт возможность строить пользовательские интерфейсы на основе компонентов, эффективно управлять состоянием приложения и работать с виртуальным DOM, что делает его идеальным инструментом для создания масштабируемых и высокопроизводительных веб-приложений.

Актуальность данной темы обусловлена растущей потребностью в современных, гибких и производительных решениях для электронной коммерции. Онлайн-магазины, разработанные с применением SPA и CSR, демонстрируют лучшие показатели вовлеченности пользователей, времени отклика и скорости

навигации. Разработка такого приложения позволяет не только повысить лояльность клиентов, но и улучшить внутреннюю структуру программного продукта за счет модульности и повторного использования компонентов.

Целью данной работы является разработка одностраничного веб-приложения для онлайн-магазина с использованием клиентского рендеринга и библиотеки React.

Для достижения этой цели необходимо решить следующие **задачи**:

- Изучить принципы работы SPA и CSR, а также возможности и преимущества использования React;
- Сформулировать технические и функциональные требования к системе;
- Разработать архитектуру и структуру проекта;
- Реализовать основные компоненты и функциональные модули онлайн-магазина;
- Обеспечить маршрутизацию между страницами и взаимодействие с внешним API;
- Провести тестирование и оптимизацию приложения.

Объектом исследования в данной работе является процесс создания одностраничных веб-приложений.

Предметом исследования – технология React и её применение в разработке SPA с использованием CSR.

Структура дипломной работы включает введение, две главы, заключение, список использованных источников и приложения. В первой главе рассматриваются теоретические аспекты разработки SPA, принципы CSR, а также обзор технологий и инструментов, применяемых при создании онлайн-магазинов. Во второй главе описываются этапы проектирования и реализации приложения: от постановки задачи до итогового тестирования готовой системы. Заключение содержит обобщение результатов и направления для дальнейшего развития проекта.

INTRODUCTION

Currently, the web development market is very demanding in terms of quality, convenience, and speed of products. This is especially true for online stores - services from which users expect not only a wide range of functionality, but also the most intuitive interface of the application. The most effective approach to creating applications of this type is single-page applications (or SPA, Single Page Application), which work on the principle of client rendering (CSR, Client-Side Rendering). This approach offers fast page loading, smooth navigation without a full reboot, and a high level of interactivity.

SPA differs from traditional multipage solutions in that it downloads the main content of the site once, after which all interaction takes place on the client side. This method significantly speeds up the interface and reduces the load on the server. CSR technology plays an important role in this context, providing dynamic data updates and content display without reloading the page. This approach is beneficial for online stores, where high response speed and constant data updates are required.

The most popular library for implementing SPA with CSR is the React JavaScript library. React provides the ability to build component-based user interfaces, effectively manage application state, and work with the virtual DOM, making it an ideal tool for creating scalable and high-performance web applications.

The relevance of this topic is due to the growing need for modern, flexible and productive e-commerce solutions. Online stores developed using SPA and CSR demonstrate the best indicators of user engagement, response time and navigation speed. The development of such an application allows not only to increase customer loyalty, but also to improve the internal structure of the software product through modularity and reuse of components.

The **purpose** of this work is to develop a single-page web application for an online store using client rendering and the React library.

To achieve this goal, it is necessary to solve the following **tasks**:

- Explore the principles of SPA and CSR, as well as the possibilities and benefits of using React;
- Formulate the technical and functional requirements for the system;
- Develop the architecture and structure of the project;
- Implement the main components and functional modules of an online store;
- Provide routing between pages and interaction with an external API;
- To test and optimize the application.

The **object** of research in this paper is the process of creating single-page web applications.

The **subject** of the research is React technology and its application in SPA development using CSR.

The structure of the thesis includes an introduction, two chapters, a conclusion, a list of sources used, and appendices. The first chapter discusses the theoretical aspects of SPA development, CSR principles, as well as an overview of technologies and tools used in creating online stores. The second chapter describes the stages of application design and implementation: from setting the task to the final testing of the finished system. The conclusion contains a summary of the results and directions for further development of the project.

ГЛАВА I. ИССЛЕДОВАНИЕ ТЕХНОЛОГИЙ ДЛЯ РАЗРАБОТКИ ОДНОСТРАНИЧНЫХ ВЕБ-ПРИЛОЖЕНИЙ

1.1 Обзор технологий для разработки SPA

Одностраничные приложения (SPA – Single Page Applications) представляют собой тип веб-приложений, в которых взаимодействие пользователя с системой осуществляется без полной перезагрузки страницы. Приложения такого типа загружаются один раз, после чего всё дальнейшее взаимодействие, включая навигацию и загрузку данных, осуществляется средствами JavaScript непосредственно в браузере пользователя. Это позволяет добиться высокой отзывчивости интерфейса и улучшенного пользовательского опыта.

Современная разработка SPA основывается на использовании клиентских JavaScript-фреймворков и библиотек. Наиболее популярными среди них являются React, Angular и Vue. Каждая из этих технологий имеет свои особенности, преимущества и области применения, но все они объединены компонентным подходом к построению интерфейса, управлением состояния и использованием виртуального DOM или его аналогов.

React – JavaScript-библиотека с открым исходным кодом. Она предоставляет разработчику удобный способ создавать интерфейсы из компонентов, которые могут повторно использоваться, комбинироваться и изменяться независимо друг от друга. React имеет активное сообщество, широкую экосистему и совместим с множеством вспомогательных библиотек для маршрутизации, управления состоянием, тестирования и др.

Angular – это полнофункциональный фреймворк от Google, который предоставляет комплексное решение “всё в одном” для построения SPA. Он включает средства для работы с формами, маршрутизацией HTTP-запросами, а также встроенные механизмы инъекций зависимостей. Angular использует

TypeScript как основной язык разработки, что способствует лучшей типизации и поддержке крупных проектов.

Vue.js – прогрессивный JavaScript-фреймворк, ориентированный на простоту и гибкость. Он сочетает в себе лучшие идеи из React и Angular, позволяя разрабатывать как простые так и масштабные приложения. Vue.js имеет развитую экосистему (Vue Router, Vuex) и поддерживает модульный подход к построению интерфейса.

Помимо основных библиотек и фреймворков, для разработки SPA необходимы и другие технологии:

- HTML5 и CSS3 – используются для разметки и стилизации интерфейса. Современные методики, такие как CSS-модули, SCSS и CSS-in-JS, упрощают работу со стилями в компонентных системах.
- JavaScript (или TypeScript) – основной язык программирования для SPA. Использование TypeScript улучшает читаемость кода и снижает количество ошибок благодаря статической типизации.
- Сборщики и транспилеры – Webpack, Vite, Babel используются для подготовки исходного кода к запуску в браузере, сборки модулей и оптимизации.
- Менеджеры пакетов – npm и yarn позволяют устанавливать и управлять зависимостями проекта.
- Системы контроля версий – Git и платформы вроде GitHub применяются для управления изменениями и совместной работы над проектом.

Также важной составляющей является работа с API – внешними источниками данных, чаще всего реализованными в виде RESTful или GraphQL-сервисов. Асинхронное взаимодействие с такими API реализуется через fetch, Axios или например специальные библиотеки, поддерживающие управление побочными эффектами (например, Redux Thunk или RTK Query).

Таким образом, разработка SPA – это комплексный процесс, включающий использование большого количества технологий, каждая из которых вносит вклад

в создание гибкого, быстрого и масштабируемого веб-приложения. В следующих разделах рассматриваются более конкретные аспекты, в том числе клиентская отрисовка, работа React и специфика онлайн-магазинов.

1.2 Клиентский рендеринг (CSR) и его применение в SPA

Клиентский рендеринг (CSR – Client-Side Rendering) представляет собой способ отображения веб-страницы, при котором формирование визуального интерфейса происходит на стороне клиента, то есть в браузере пользователя. В отличие от серверного рендеринга, где HTML-код страницы полностью формируется на сервере и отправляется клиенту в готовом виде, при CSR браузеру передаётся минимальный HTML-шаблон и JavaScript-код, который уже на клиенте обрабатывает данные и динамически формирует пользовательский интерфейс.

Основная идея клиентского рендеринга заключается в том, чтобы разделить ответственность между сервером и клиентом: сервер занимается только выдачей данных (например, в JSON), а всё отображение и логика работы пользовательского интерфейса выполняются в браузере. Это позволяет создать более отзывчивое и динамичное взаимодействие между пользователем и системой.

Для SPA клиентский рендеринг является базовым и наиболее часто используемым подходом. После первоначальной загрузки страницы приложение “живёт” в браузере: пользователь может переходить между страницами, загружать новые данные, изменять состояние интерфейса – и всё это происходит без перезагрузки страницы. Навигация реализуется средствами JavaScript и библиотек маршрутизации, а получение данных осуществляется через асинхронные запросы к серверу.

Преимущества CSR в контексте SPA включают:

- Быстрая и плавная навигация – переходы между разделами сайта происходят мгновенно, без повторной загрузки HTML.

- Уменьшение нагрузки на сервер – сервер обрабатывает только запросы к API, не генерируя HTML-страницы.
- Разделение логики интерфейса и логики данных – это упрощает масштабирование и сопровождение приложения.
- Гибкость и интерактивность – пользовательский интерфейс легко адаптируется под действия пользователя и может реагировать в реальном времени.

Однако клиентский рендеринг имеет и определённые недостатки:

- Медленная первая загрузка (First Load) – так как интерфейс формируется только после загрузки и выполнения JavaScript, это может занять некоторое время.
- Проблемы с индексацией страниц – поисковые системы могут не корректно индексировать контент, формируемый на клиенте, что снижает эффективность SEO (эта проблема частично решается с помощью пререндеринга или гибридных подходов)
- Зависимость от производительности клиента – при слабом устройстве у пользователя может снижаться отзывчивость интерфейса.

В современных условиях эти недостатки частично нивелируются. Разработчики используют методы оптимизации, такие как code splitting, lazy loading, и асинхронная загрузка компонентов. Кроме того, при необходимости можно применять гибридные подходы – например, использовать серверный рендеринг (SSR) для первых загрузок, а затем переходить на CSR в рамках всего остального взаимодействия.

В контексте онлайн-магазинов CSR позволяет реализовать удобную и быструю работу с каталогом товаров, корзиной, фильтрами и другими элементами интерфейса, минимизируя время отклика. Пользователь может добавлять товары в корзину, переключать категории, сортировать и фильтровать продукты без ощущения задержки или перезагрузки страницы, что напрямую влияет на положительный пользовательский опыт и увеличивает вероятность завершения покупки.

Таким образом, клиентский рендеринг является фундаментальной технологией для построения современных одностраничных приложений, обеспечивая высокую интерактивность, гибкость и удобство пользования. При грамотной реализации и оптимизации его использование особенно эффективно в проектах электронной коммерции.

1.3 Использование React для разработки в SPA

Как уже говорилось ранее, React – это библиотека JavaScript с открытым исходным кодом, предназначенная для построения пользовательских интерфейсов, прежде всего одностраничных приложений (SPA), и сегодня является одной из наиболее популярных и востребованных технологий во фронтенд-разработке.

Основной идеей React является компонентный подход: всё приложение разбивается на независимые, переиспользуемые блоки – компоненты. Каждый компонент отвечает за конкретную часть интерфейса и может содержать как логику, так и отображение. Такой подход делает код более модульным, поддерживаемым и масштабируемым.

Ключевые особенности React, делающие его особенно удобным для разработки SPA:

- JSX – расширение синтаксиса JavaScript, позволяющее писать HTML-подобный код внутри JavaScript. Это упрощает построение структуры интерфейса и делает код более читабельным.
- Virtual DOM – виртуальное представление реального DOM, позволяющее эффективно определять минимальные изменения в интерфейсе и применять их без полной перерисовки. Это значительно повышает производительность.
- Односторонний поток данных – управление данными в React осуществляется сверху вниз (от родителя к дочерним компонентам), что делает поведение приложения предсказуемым.

- Hooks – функции, такие как useState, useEffect, useContext, и другие, которые позволяют использовать состояние и другие возможности React без написания классов. Они стали стандартом разработки с появлением React 16.8.

Для реализации полноценного SPA на базе React дополнительно используются внешние библиотеки и инструменты:

- React Router – библиотека для организации маршрутизации в одностраничных приложениях. Она позволяет определять пути (routes), связывать их с компонентами и переключаться между “страницами” без перезагрузки.
- Redux – инструмент для централизованного управления состоянием приложения. Полезен в приложениях с большим количеством взаимосвязанных компонентов и сложной логикой.
- Axious или fetch API – используются для отправки HTTP-запросов к серверу и загрузки данных.
- Context API – встроенный инструмент для передачи данных через дерево компонентов без необходимости пробрасывания props на каждом уровне.
- React Query, Zustand, Recoil и RTK Query – альтернативы Redux, которые упрощают работу с состоянием и асинхронными запросами.

React активно используется для создания интерфейсов онлайн-магазинов благодаря своей гибкости, высокой производительности и широкому сообществу.

В e-commerce-проектах с помощью React удобно реализовать:

- каталог товаров с фильтрами и сортировкой;
- детальные страницы товаров;
- корзину с добавлением и удалением товаров;
- формы оформления заказа;
- регистрацию и авторизацию пользователей;
- адаптивный дизайн для разных устройств.

Кроме того, React поддерживает интеграцию с различными UI-библиотеками (например, Material UI, Ant Design), что ускоряет разработку за счет готовых компонентов и улучшает визуальное оформление интерфейса.

Важно отметить, что благодаря своей архитектуре React хорошо масштабируется – приложение может постепенно развиваться от простого прототипа до полнофункционального сервиса с множеством страниц и сложной бизнес-логикой, не теряя при этом управляемости кода.

Таким образом, React предоставляет все необходимые инструменты для построения современных SPA. Его широкие возможности, активное сообщество и богатая экосистема делают его идеальным выбором для разработки онлайн-магазинов и других высокоинтерактивных веб-приложений.

1.4 Особенности архитектуры онлайн-магазина

Разработка онлайн-магазина как веб-приложения требует особого внимания к архитектуре, поскольку он включает в себя множество функциональных модулей, взаимодействующих друг с другом. Структура такого приложения должна быть гибкой, масштабируемой и легко поддерживаемой, особенно в условиях роста ассортимента товаров, пользовательской базы и бизнес-логики.

Архитектура онлайн-магазина, реализованного в виде SPA, строится на основе компонентного подхода. Это позволяет разделить приложение на логические части (каталог, карточка товара, корзина, оформление заказа, авторизация и др.), каждая из которых реализуется как независимый компонент или набор компонентов. Все эти части объединяются в единое приложение посредством маршрутизации и управления состоянием.

Онлайн-магазин, как правило, включает следующие ключевые функциональные блоки:

- Главная страница – содержит информацию о популярных товарах, акциях, категориях, новинках. Это первая точка взаимодействия пользователя с приложением.
- Каталог товаров – страница со списком товаров, возможностью фильтрации и сортировки по различным параметрам (цена, категория, популярность, наличие и т. д.).
- Карточка товара – отдельный компонент, отображающий детальную информацию о выбранном товаре, включая фотографии, описание, характеристики, отзывы, цену и возможность добавления в корзину.
- Корзина – отдельный модуль, хранящий информацию о выбранных товарах, количестве, сумме заказа и возможностью удаления или изменения количества.
- Оформления заказа – включает формы ввода контактной информации, адреса доставки, выбора способа оплаты.
- Регистрация и авторизация – обеспечивает защиту персональных данных и дают возможность пользователю просматривать историю заказов, настраивать профиль и др.
- Личный кабинет – страница для зарегистрированных пользователей, включающая управление профилем, просмотр заказов, настройки уведомлений и т. д.

Все эти блоки должны быть тесно связаны с системой управления состоянием. Для этого обычно используется Redux или его аналоги, позволяющие централизованно хранить данные о текущем пользователе, состоянии корзины, товарах и т. д.

Кроме того, архитектура онлайн-магазина должна обеспечивать следующие аспекты:

- Работы с API – приложение должно уметь отправлять запросы к серверу (например, для получения списка товаров, оформления заказа, регистрации пользователя), обрабатывать ответы и отображать данные пользователю. Для этого, как правило, используются REST API или GraphQL.

- Асинхронность – обработка данных в реальном времени, отображение индикаторов загрузки, задержки, ошибки и повторные запросы – важная часть UX в e-commerce-приложении.
- Аутентификация и авторизация – реализуются через токены (например, JWT), хранящиеся в браузере и автоматически передающиеся при последующих запросах.
- Валидация данных – как на клиентской, так и на серверной стороне, чтобы обеспечить корректность введённой пользователем информации (адрес, email, номер телефона и др.)
- Маршрутизация – реализуется с помощью React Router и позволяет пользователю переходить между страницами без перезагрузки.

Важной особенностью архитектуры онлайн-магазина является адаптивность интерфейса. Пользователи часто заходят с мобильных устройств, поэтому интерфейс должен быть адаптирован под различные разрешения экранов.

Также необходимо учитывать возможность масштабирования. По мере развития проекта могут появляться новые модули (например, система скидок, платёжные шлюзы, отзывы, рейтинг товаров), поэтому архитектура должна предусматривать возможность интеграции нового функционала без полной переработки существующего кода.

Таким образом, архитектура онлайн-магазина на базе SPA должна быть тщательно продумана: обеспечивать разделение логики по слоям, повторное использование компонентов, централизованное управление данными и простоту интеграции новых функций. Это позволяет создать удобный, быстрый и масштабируемый продукт, соответствующий современным требованиям электронной коммерции.

1.5 Современные подходы к созданию интерфейсов для онлайн-магазина

Пользовательский интерфейс (UI) играет решающую роль в успехе онлайн-магазина. От удобства навигации, ясности структуры и эстетики внешнего вида напрямую зависят такие ключевые показатели, как вовлеченность пользователей, глубина просмотра, конверсия и уровень возврата клиентов. Современные подходы к созданию интерфейсов e-commerce приложений стремятся объединить визуальную привлекательность с максимальной функциональностью и доступностью.

Одним из ключевых принципов современного UI-дизайна является ориентированность на пользователя (user-centered design). Интерфейс должен быть интуитивно понятным, адаптированным к ожиданиям и привычкам целевой аудитории. Это проявляется в использовании знакомых шаблонов поведения, понятных иконок, минималистичного оформления и логичного размещения элементов управления.

Другим важным направлением является адаптивный и отзывчивый дизайн (responsive design), обеспечивающий корректное отображение и функционирование интерфейса на любых устройствах – от настольных компьютеров до мобильных телефонов. Для этого применяются медиазапросы CSS, флекс-сетки, а также фреймворки, такие как Tailwind CSS или Bootstrap.

Современные интерфейсы онлайн-магазинов также активно используют асинхронные механизмы взаимодействия с пользователем, реализуемые через JavaScript и React. Это позволяет реализовать “живой” интерфейс: товары добавляются в корзину без перезагрузки страницы, данные обновляются динамически, фильтры работают в реальном времени. Такие механики повышают интерактивность и делают взаимодействие более плавным и естественным.

Для повышения эффективности взаимодействия также применяются:

- Поисковые строки с автодополнением;
- Умные фильтры и сортировка по популярным параметрам;

- Визуальные подсказки и всплывающие уведомления (например, об успешном добавлении товара в корзину);
- Ленивая загрузка изображений и бесконечный скроллинг;
- Анимации и переходы, улучшающие восприятие действий пользователя.

Важную роль в интерфейсе играет UI-библиотека компонентов. Такие библиотеки, как Material UI, Ant Design или Chakra UI, Позволяют использовать готовые адаптированные элементы интерфейса: кнопки, поля ввода, модальные окна, табы, аккордеоны и т. д. Это ускоряет разработку, снижает количество ошибок и обеспечивает единый визуальный стиль. Использование таких библиотек также обеспечивает доступность, что делает приложение более универсальным и соответствующим современным стандартам.

Неотъемлемой частью интерфейса онлайн-магазина является визуализация информации о товарах: карточки с изображениями, ценой, рейтингом и кнопкой добавления в корзину. Визуальное представление должно быть чистым, не перегруженным и позволять пользователю быстро ориентироваться в ассортименте. Для этого используются гриды, акценты на ключевые элементы и продуманные схемы цветовых контрастов.

Также важно реализовать единый пользовательский поток – от выбора товара до оформления заказа. На каждом этапе пользователь должен понимать, где он находится, что нужно сделать дальше, и как вернуться к предыдущему шагу. В этом помогают индикаторы прогресса, подтверждающие сообщения и логично выстроенные маршруты внутри приложения.

Отдельное внимание уделяется скорости загрузки и отзывчивости интерфейса. Даже незначительные задержки могут привести к отказу от покупки. Поэтому применяется оптимизация: уменьшение количества запросов, кеширование, отложенная загрузка (lazy loading), использование CDN для изображений и скриптов.

В последние годы также растёт интерес к персонализации интерфейса. На основе истории покупок, поведения пользователя или геолокации можно показывать релевантные товары, подстраивать рекомендации, адаптировать

главную страницу. Это увеличивает вовлечённость и стимулирует повторные покупки.

Таким образом, современные подходы к созданию интерфейсов для онлайн-магазинов основаны на гибкости, адаптивности, удобстве и визуальной чистоте. В сочетании с технологической основой на базе React и SPA-архитектуры эти подходы позволяют создавать эффективные, конкурентоспособные и удобные веб-приложения, соответствующие ожиданиям пользователей.

ГЛАВА II. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ОДНОСТРАНИЧНОГО ПРИЛОЖЕНИЯ

2.1 Разработка технического задания и требований

На этапе начала разработки программного продукта важнейшим шагом является формирование технического задания (ТЗ), в котором подробно описываются цели, функции, архитектура и ключевые требования к создаваемому приложению. Четко определённое ТЗ служит основой всей дальнейшей разработки, помогает избежать двусмысленностей и обеспечивает соответствие конечного результата ожиданиям пользователя.

Цель проекта – разработка одностраничного веб-приложения онлайн-магазина с использованием библиотеки React и технологии клиентского рендеринга (CSR), обеспечивающего удобный и быстрый пользовательский интерфейс для просмотра, выбора и заказа товаров.

Приложение должно решать следующие основные задачи:

- Предоставление пользователю удобного каталога товаров;
- Возможность фильтрации, сортировки и поиска товаров;
- Отображение подробной информации о каждом товаре;
- Добавление и удаление товаров из корзины;
- Оформление заказа через форму ввода данных;
- Регистрация и вход в систему с сохранением пользовательских данных;
- Адаптация интерфейса под различные устройства.

Для достижения поставленной цели формулируются функциональные требования.

Каталог товаров:

- Отображение списка товаров с основными характеристиками (название, изображение, цена);
- Возможность сортировки по цене, названию и другим параметрам;

- Фильтрация по категориям;
- Динамическая подгрузка товаров при прокрутке или переключении страниц.

Карточка товара:

- Отображение полной информации о товаре: описание, изображения, характеристики, цена;
- Возможность добавления товара в корзину.

Корзина:

- Просмотр добавленных товаров;
- Изменение количества товаров;
- Удаление товаров из корзины;
- Отображение общей стоимости заказа.

Оформление заказа:

- Форма ввода контактных данных (имя, телефон, email, адрес доставки);
- Выбор способа доставки и оплаты;
- Отправка заказа на сервер и отображение подтверждения.

Пользовательская аутентификация:

- Регистрация новых пользователей;
- Авторизация с проверкой введённых данных;
- Сохранение состояния корзины и истории заказов для авторизованных пользователей.

Адаптивный дизайн:

- Корректное отображение интерфейса на устройствах с различным разрешением экрана;
- Отзывчивость элементов управления и верстки.

Интерактивность и отзывчивость интерфейса:

- Добавление и удаление товаров из корзины без перезагрузки страницы;
- Отображение уведомлений и подтверждений действий пользователя.

Нефункциональные требования включают:

- Высокая скорость загрузки и отклика интерфейса;
- Чистота и читаемость кода;

- Модульность и расширяемость архитектуры;
- Минимизация повторного рендеринга компонентов;
- Использование актуальных и поддерживаемых библиотек и технологий;
- Поддержка последних версий популярных браузеров.

Технологии и инструменты, используемые в проекте:

- Язык программирования: JavaScript (ES6+), частично TypeScript;
- Библиотека: React;
- Управление маршрутизацией: React Router;
- Управление состоянием: Redux или Context API;
- HTTP-запросы: Axious или fetch API;
- Стилизация: CSS Modules / Styled Components / Tailwind CSS;
- Сборка проекта: Vite или Create React App;
- Хранение пользовательских данных: localStorage (для неавторизованных пользователей);
- Инструменты контроля качества: ESLint, Prettier;
- Система контроля версий: GIT.

Таким образом, сформулированное техническое задание определяет функциональные границы проекта, архитектурные ориентиры и технологическую основу будущего приложения. На следующем этапе осуществляется проектирование архитектуры онлайн-магазина с учётом всех вышеуказанных требований.

2.2 Проектирование архитектуры приложения

Этап проектирования архитектуры является одним из ключевых при разработке любого программного продукта, особенно – для одностраничных приложений с богатой функциональностью, таких как онлайн-магазины. От качества архитектурных решений напрямую зависят масштабируемость,

поддерживаемость, читаемость и расширяемость кода, а также удобство взаимодействия пользователей с приложением.

Приложение онлайн-магазина, реализованного в виде SPA с использованием React, строится по компонентно-модульному принципу. Архитектура такого приложения должна учитывать разбиение на логические блоки, поддержку маршрутизации, централизованное управление состоянием и взаимодействие с серверной частью.

Общая структура проекта

Проект условно делится на следующие основные директории:

- /components – переиспользуемые визуальные компоненты интерфейса: кнопки, карточки товаров, поля ввода, модальные окна и др.
- /pages – компоненты страниц, которые соответствуют маршрутам: Главная, Каталог, Товар, Корзина, Оформление заказа, Личный кабинет и др.
- /store – глобальное хранилище состояния приложения (например, Redux или Context API)
- /services – модули для работы с внешними API: запросы на получение списка товаров, добавление заказов, авторизация и регистрация.
- /utils – вспомогательные функции, утилиты и константы.
- /styles – глобальные и модульные стили.
- /router – конфигурация маршрутов с использованием React Router.

Такое разделение позволяет изолировать ответственность между слоями и упростить навигацию по коду.

Компонентный подход

Каждая часть интерфейса разрабатывается в виде самостоятельного компонента, который может быть повторно использован и протестирован.

Примеры компонентов:

- ProductCard – карточка товара в списке;
- ProductDetails – подробная информация о товаре;
- CartItem – товар в корзине;
- InputField, Button, Modal – элементы интерфейса общего назначения.

Реализация маршрутизации

Для обеспечения переходов между страницами без перезагрузки используется библиотека React Router. Примеры маршрутов:

- / – главная страница;
- /products – каталог товаров;
- /products/:id – страница конкретного товара;
- /cart – корзина;
- /checkout – оформление заказа;
- /login и /register – формы входа и регистрации;
- /profile – личный кабинет пользователя.

Каждому маршруту соответствует компонент страницы, который отображается в зависимости от URL.

Управление состоянием

Для хранения данных, доступных во всём приложении (состояние корзины, пользователь, авторизация, список товаров), использует глобальное хранилище:

- Redux Toolkit или Context API – для организации хранилища;
- Redux Thunk или RTK Query – для обработки асинхронных запросов;
- Selectors и actions – для получения данных и вызовов изменений состояния.

Состояние разбивается на срезы: cartSlice, userSlice, productSlice, каждый из которых содержит логику управления конкретной частью данных.

Работа с API

Для взаимодействия с сервером выделяются сервисные функции (например, api/products.js, api/auth.js). Все запросы (GET, POST, PUT, DELETE) реализуются с помощью библиотеки Axious или встроенного fetch.

Примеры функций:

- Получение списка товаров;
- Получение данных одного товара по ID;
- Отправка заказа;
- Регистрация и вход пользователя.

Хранение данных

Для хранения промежуточных данных (например, корзины у неавторизованных пользователей) применяется localStorage. При авторизации пользовательская информация и корзина синхронизируется с сервером.

Принципы адаптивности и повторного использования

Компоненты создаются с учётом возможности повторного использования и адаптации под разные устройства. Используются:

- Flexbox / Grid-сетки;
- Медиа-запросы;
- Мобильное меню и выпадающие панели;
- Библиотеки компонентов (например, Material UI или Tailwind CSS)

Таким образом, архитектура приложения построена по принципам модульности, повторного использования, централизованного управления данными и поддержки масштабирования. Следующий шаг – реализации ключевых компонентов приложения, что и будет рассмотрено в следующем разделе.

2.3 Разработка компонентов приложения

Разработка пользовательских компонентов – один из ключевых компонентов создания одностраничного приложения. Компоненты в React представляют собой независимые, переиспользуемые элементы интерфейса, каждый из которых инкапсулирует свою структуру, логику и стили. Такой подход позволяет разрабатывать приложение по частям, облегчает отладку и поддержку, а также повышает масштабируемость проекта.

В приложении онлайн-магазина компоненты можно условно разделить на несколько категорий:

Компоненты общего назначения

Эти компоненты используются во многих частях приложения и не зависят от конкретного бизнес-контекста.

- Button – кнопка с настраиваемыми размерами, цветами, иконками;
- InputField – текстовое поле с валидацией и обработкой ошибок;
- Modal – всплывающее окно для отображения информации или подтверждений;
- Loader – индикатор загрузки, отображаемый при ожидании данных.

Компоненты каталога товара

Эти компоненты обеспечивают отображение списка товаров, карточек и взаимодействия с пользователем.

- ProductCard – карточка товара, отображающая изображение, название, цену и кнопку “в корзину”;
- ProductList – список товаров, формируемый из массива данных;
- ProductFilter – панель фильтрации по категории, цене, наличию и другим параметрам;
- ProductSort – элемент управления сортировкой (по цене, алфавиту и т. д.);
- SearchBar – строка поиска по ключевым словам.

Компоненты страницы товара

Для отображения полной информации о товаре реализуется отдельный компонент:

- ProductDetails – отображает подробности: большое изображение, полное описание, характеристики, отзывы, кнопка добавления в корзину, счётчик количества.

Компоненты корзины

Корзина – одна из важнейших частей любого онлайн-магазина. Она включает:

- CartItem – блок, отображающий отдельный товар в корзине (название, цена, количество, кнопки управления);
- CartList – список всех товаров в корзине;
- CartSummary – итоговая сумма заказа, кнопка перехода к оформлению;

- EmptyCart – сообщение, отображаемое при отсутствии товаров.

Корзина поддерживает динамическое изменение количества, удаление товаров и автоматическое обновление общей стоимости.

Компоненты оформления заказа

Этап оформления заказа реализуется в виде формы:

- CheckoutForm – включает поля ввода имени, телефона, адреса, выбора доставки и оплаты;
- OrderSummary – отображение итогов заказа перед подтверждением;
- OrderConfirmation – страница или модальное окно с подтверждением оформления заказа.

Все поля формы проходят клиентскую валидацию, а после отправки данные передаются на сервер.

Компоненты авторизации и личного кабинета

Для работы с пользователями создаются отдельные страницы и компоненты

- LoginForm и RegisterForm – формы входа и регистрации, с обработкой ошибок и валидацией;
- UserProfile – страница профиля пользователя с возможностью изменения данных;
- OrderHistory – отображение списка предыдущих заказов.

Авторизованные пользователи получают доступ к расширенному функционалу, включая сохранение истории заказов и настроек.

Компоненты навигации и макета

Взаимодействие между страницами и общая структура интерфейса реализуются с помощью следующих компонентов:

- Navbar – верхняя панель с логотипом, ссылками на основные страницы и иконкой корзины;
- Footer – нижний колонтитул с контактной информацией, ссылками и дополнительным контентом;
- Sidebar – боковое меню с фильтрами (для каталога);
- Layout – обёртка для всех страниц, включая общие элементы интерфейса.

Все компоненты оформлены с использованием адаптивной верстки и обеспечивают корректную работу на различных устройствах. Стилизация реализуется с помощью CSS-модулей, Tailwind CSS или Styled Components, в зависимости от выбранной методики.

Кроме того, при разработке компонентов учитываются:

- Обработка ошибок (например, при загрузке товаров);
- Отображение состояний загрузки (isLoading);
- Взаимодействие с глобальным состоянием (например, корзина хранится в Redux или Context API);
- Взаимодействие с API (например, при добавлении товара в корзину или авторизации).

Таким образом, компонентная структура приложения позволяет добиться высокой степени переиспользуемости, упрощает разработку и поддержку интерфейса, а также способствует лучшей организации кода. В следующем разделе будет рассмотрен процесс настройки маршрутизации и навигации между компонентами.

2.4 Тестирование системы

Одной из ключевых особенностей одностраничного приложения (SPA) является возможность навигации между разделами без перезагрузки страницы. Для этого в React-приложении используется библиотека React Router, которая обеспечивает гибкую маршрутизацию, поддержку параметров в URL, вложенных маршрутов и защиты маршрутов (route guarding).

Кроме маршрутизации, важным элементом архитектуры является взаимодействие с серверной частью через HTTP-запросы, обеспечивающие загрузку и отправку данных. Современные подходы предполагают использование асинхронных запросов с помощью Axious или встроенного fetch, а также обработку состояний загрузки, ошибок и успешных ответов.

Маршрутизация с помощью React Router

В проекте используется React Router версии 6. В таблице 1.1 представлены основные маршруты и их назначение.

Таблица 1.1 Основные маршруты и их назначение.

URL	Компонент	Назначение
/	HomePage	Главная страница
/products	ProductCatalogPage	Каталог товаров
/product/:id	ProductDetailsPage	Страница товара по ID
/cart	CartPage	Корзина
/checkout	CheckoutPage	Оформление заказа
/login, /register	LoginPage, RegisterPage	Авторизация и регистрация
/profile	ProfilePage	Личный кабинет
*	NotFoundPage	Страница 404

Маршруты определяются в компоненте AppRoutes и оборачиваются в компонент <BrowserRouter>. Переход между страницами осуществляется с помощью компонента <Link> или функции useNavigate().

Также реализуются защищённые маршруты – доступ к личному кабинету или оформлению заказа возможен только при наличии авторизации. Это обеспечивает компонентом-обёрткой PrivateRoute, который проверяет состояние пользователя из глобального хранилища.

Взаимодействие с сервером

Взаимодействие с серверной частью осуществляется через API-запросы. Для организации этого взаимодействия создаются отдельные модули в папке /services, например:

- productService.js – запросы к товарам;

- cartService.js – работа с корзиной;
- authService.js – регистрация, вход, выход;
- orderService.js – оформление и получение заказов.

Пример функции получения списка товаров с помощью Axios вы можете видеть на рисунке 2.1.

```
import axios from 'axios';

export const fetchProducts = async () => {
  const response = await axios.get('/api/products');
  return response.data;
};
```

Рисунок 2.1. Пример функции получения списка товаров.

Пример запроса оформления заказа вы можете видеть на рисунке 2.2.

```
export const submitOrder = async (orderData) => {
  const response = await axios.post('/api/orders', orderData);
  return response.data;
};
```

Рисунок 2.2. Пример запроса оформления заказов.

При отправке запросов учитываются:

- загрузка (isLoading) – отображаются спиннеры;
- ошибки – отображаются уведомления об ошибке;
- успешное завершение – показывается сообщение об успешной операции.

Асинхронная логика и состояние

Асинхронная логика реализуется через:

- хуки useEffect для загрузки данных при монтировании компонента;
- диспетчеризацию действий Redux с использованием Redux Thunk;

- кастомные хуки для повторного использования логики (например, useProducts, useAuth).

Пример загрузки товаров вы можете видеть на рисунке 2.3.

```
useEffect(() => {
  dispatch(fetchProductsAsync());
}, []);
```

Рисунок 2.3. Пример загрузки товаров.

Безопасность запросов

При авторизации используются JWT-токены, сохраняемые в localStorage или sessionStorage, и автоматически добавляемые в заголовки запросов. Процесс этого вы можете видеть на рисунке 2.4.

```
axios.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
```

Рисунок 2.4. Процесс сохранения JWT-токенов.

Таким образом, маршрутизация и взаимодействие с сервером в React-приложении организованы по современным стандартам: приложение обеспечивает быструю навигацию, надёжную загрузку данных и безопасную работу с пользовательскими запросами. В следующем разделе будет рассмотрено управление состоянием с помощью Redux.

2.5 Реализация системы управления состоянием с использованием Redux

При создании масштабируемого и функционально насыщенного SPA крайне важно эффективно управлять состоянием приложения. В онлайн-магазине необходимо хранить данные о товарах, содержимом корзины, состоянии пользователя, заказах и других элементах, которые используются в различных частях интерфейса. Для этого используется библиотека Redux – один из самых популярных инструментов для централизованного управления состоянием в React-приложениях.

Принципы работы Redux

Redux основывается на трёх ключевых принципах:

1. Единый источник правды – всё состояние приложения хранится в одном глобальном хранилище (store).
2. Состояние доступно только для чтения – его можно изменить только с помощью специальных действий (actions).
3. Изменения происходят с помощью чистых функций – редьюсеров (reducers), которые принимают текущее состояние и действие, возвращая новое состояние.

Это обеспечивает предсказуемость поведения, удобство отладки, возможность использования инструментов разработчика Redux DevTools и масштабируемость.

Установка и настройка Redux Toolkit

Для упрощения работы используется Redux Toolkit – официальная надстройка, упрощающая создания хранилища, редьюсеров и асинхронной логики: npm install @reduxjs/toolkit react-redux

На рисунке 2.5 вы можете увидеть каким образом создаётся хранилище.

```
// store.js
import { configureStore } from '@reduxjs/toolkit';
import cartReducer from './slices/cartSlice';
import productsReducer from './slices/productsSlice';
import userReducer from './slices/userSlice';

export const store = configureStore({
  reducer: {
    cart: cartReducer,
    products: productsReducer,
    user: userReducer,
  },
});
```

Рисунок 2.5. Процесс создания хранилища.

На рисунке 2.6. вы можете увидеть как хранилище подключается к приложению через <Provider>.

```
import { Provider } from 'react-redux';
import { store } from './store';

<Provider store={store}>
  <App />
</Provider>
```

Рисунок 2.6. Процесс подключения хранилища к приложению.

Создания срезов состояния (slices)

Каждая логическая часть приложения (корзина, товары, пользователь) реализуется как отдельный срез (slice).

Пример среза для корзины вы можете увидеть на рисунке 2.7.

```
// cartSlice.js
import { createSlice } from '@reduxjs/toolkit';

const cartSlice = createSlice({
  name: 'cart',
  initialState: {
    items: [],
  },
  reducers: {
    addToCart(state, action) {
      state.items.push(action.payload);
    },
    removeFromCart(state, action) {
      state.items = state.items.filter(item => item.id !== action.payload);
    },
    clearCart(state) {
      state.items = [];
    },
  },
});

export const { addToCart, removeFromCart, clearCart } = cartSlice.actions;
export default cartSlice.reducer;
```

Рисунок 2.7. Пример среза для корзины.

Асинхронные запросы с createAsyncThunk

Для загрузки данных сервера используется createAsyncThunk, упрощающий работу с промисами и состояниями (pending, fulfilled, rejected). Пример его работы вы можете увидеть на рисунке 2.8.

```
// productsSlice.js
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from 'axios';

export const fetchProducts = createAsyncThunk(
  'products/fetchProducts',
  async () => {
    const response = await axios.get('/api/products');
    return response.data;
  }
);

const productsSlice = createSlice({
  name: 'products',
  initialState: { items: [], status: 'idle' },
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchProducts.pending, (state) => {
        state.status = 'loading';
      })
      .addCase(fetchProducts.fulfilled, (state, action) => {
        state.items = action.payload;
        state.status = 'succeeded';
      })
      .addCase(fetchProducts.rejected, (state) => {
        state.status = 'failed';
      });
  },
});
```

Рисунок 2.8. Пример работы createAsyncThunk

Работы с хранилищем в компонентах

Доступ к состоянию и действиям осуществляется через хуки. Пример кода вы можете увидеть на рисунке 2.9.

```
import { useSelector, useDispatch } from 'react-redux';
import { addToCart } from '../store/slices/cartSlice';

const ProductCard = ({ product }) => {
  const dispatch = useDispatch();

  const handleAdd = () => {
    dispatch(addToCart(product));
  };

  return (
    <button onClick={handleAdd}>Добавить в корзину</button>
  );
};
```

Рисунок 2.9. Пример использования хуков для доступа к состоянию.

Селекторы позволяют получить данные из хранилища: const cartItems = useSelector((state) => state.cart.items);

Персистентность состояния

Чтобы сохранить состояние корзины при обновлении страницы, используется localStorage. Отрезок кода вы можете увидеть на рисунке 2.10.

```
useEffect(() => {
  localStorage.setItem('cart', JSON.stringify(cartItems));
}, [cartItems]);
```

Рисунок 2.10. Пример сохранения состояния корзины при помощи localStorage.

Таким образом, использование Redux позволяет централизованно управлять всеми аспектами состояния приложения. Это делает код предсказуемым, упрощает отладку и облегчает взаимодействие между компонентами. Следующим этапом станет интеграция сторонних сервсисов – например, платёжных или авторизационных API.

2.6 Тестирование приложения

Тестирование – важный этап в процессе разработки любого программного продукта, обеспечивающий стабильную и предсказуемую работу приложения. В контексте веб-разработки и, в частности, одностраничных приложений на базе React, тестирование компонентов, логики и пользовательских сценариев позволяет выявить ошибки на ранних этапах и гарантировать корректную работу ключевых функций.

Тестирование React-приложения можно условно разделить на три уровня:

- Модульное (юнит-тестирование) – проверка отдельных функций и компонентов
- Интеграционное – проверка взаимодействий между несколькими модулями или компонентами;
- End-to-End (E2E) – симуляция действительно реального пользователя в интерфейсе.

Используемые инструменты

Для тестирования приложения применяются следующие библиотеки:

- Jest – тестовый фреймворк по умолчанию в React-проектах. Предоставляет возможность писать и запускать юни- и интеграционные тесты.
- React Testing Library (RTL) – инструменты для тестирования компонентов React с упором на поведение пользователя, а не реализацию.
- Cypress (опционально) – для E2E-тестирования через браузер (можно использовать на продвинутом этапе разработки).

Пример юнит-теста компонента

На рисунке 2.11. вы можете видеть пример тестирования компонента Button, отображающего текст и вызывающего функцию при клике.

```
import { render, screen, fireEvent } from '@testing-library/react';
import Button from './Button';

test('отображает переданный текст и реагирует на нажатие', () => {
  const onClick = jest.fn();
  render(<Button onClick={onClick}>Купить</Button>);

  const buttonElement = screen.getByText('Купить');
  expect(buttonElement).toBeInTheDocument();

  fireEvent.click(buttonElement);
  expect(onClick).toHaveBeenCalledTimes(1);
});
```

Рисунок 2.11. Пример тестирования компонента Button.

Тестирование компонента со взаимодействием

Пример тестирования компонента CartItem, отображающего товар и позволяющего удалить его из корзины изображен на рисунке 2.12.

```
test('удаляет товар при нажатии на кнопку', () => {
  const mockRemove = jest.fn();
  render(<CartItem product={{ title: 'Товар 1' }} onRemove={mockRemove} />

  fireEvent.click(screen.getByRole('button', { name: /удалить/i }));
  expect(mockRemove).toHaveBeenCalled();
});
```

Рисунок 2.12. Пример тестирования компонента CartItem

Проверка асинхронных операций

Пример теста компонента, который загружает список товаров из API вы можете видеть на рисунке 2.13.

```
import { fetchProducts } from './productsSlice';
import { configureStore } from '@reduxjs/toolkit';
import productsReducer from './productsSlice';

test('fetchProducts загружает данные', async () => {
  const store = configureStore({ reducer: { products: productsReducer } })
  await store.dispatch(fetchProducts());
  const state = store.getState().products;

  expect(state.items.length).toBeGreaterThan(0);
  expect(state.status).toBe('succeeded');
});
```

Рисунок 2.13. Пример теста загрузчика товаров из API.

Интеграционные тесты

Интеграционные тесты позволяют проверить сценарии перехода между страницами, добавления товара в корзину, оформления заказа. Пример простого теста маршрутизации изображен на рисунке 2.14..

```
import { MemoryRouter } from 'react-router-dom';
import App from './App';

test('переход по маршруту /cart отображает корзину', () => {
  render(
    <MemoryRouter initialEntries={['/cart']}>
      <App />
    </MemoryRouter>
  );

  expect(screen.getByText(/ваша корзина/i)).toBeInTheDocument();
});
```

Рисунок 2.14. Пример теста маршрутизации.

End-to-End тестирование

На поздних этапах разработки можно использовать инструменты E2E-тестирования, такие как Cypress. Это позволяет запускать приложения в браузере и автоматически выполнять действия: переход по страницам, ввод текста, оформление заказа. Такой подход помогает выявить ошибки, связанные с взаимодействием различных компонентов, маршрутизацией и интеграцией с сервером.

Покрытия тестами

Для анализа качества тестов используется покрытие кода (code coverage). Jest предоставляет встроенный инструмент покрытия. Команда: npm test – –coverage. Позволяет получить отчёт о том, какая часть кода покрыта тестами, и какие участки следует доработать.

Таким образом, тестирование является неотъемлемой частью разработки и обеспечивает стабильную работу всех функций онлайн-магазина. Применение модульных, интеграционных и пользовательских тестов позволяет выявлять ошибки на ранних стадиях и поддерживать высокое качество кода при масштабировании проекта.

2.7 Оптимизация производительности

Одним из ключевых факторов, влияющих на качество пользовательского опыта в веб-приложениях, является производительность. Для онлайн-магазина, реализованного как одностраничное приложение (SPA), это особенно важно: пользователи ожидают быстрой загрузки, мгновенного отклика и стабильной работы независимо от устройства или скорости соединения. В данном разделе рассматриваются методы и подходы к оптимизации производительности React-приложения.

Code splitting (разделение кода)

React-приложение при сборке объединяется в один JavaScript-файл, который может быть довольно большим. Чтобы сократить время первой загрузки

страницы, используется разделение кода, при котором приложение разбивается на отдельные части (чанки), подгружаемые по мере необходимости.

Для этого применяется React.lazy() и Suspense. Пример применения вы можете увидеть на рисунке 2.15.

```
import React, { lazy, Suspense } from 'react';

const ProductDetailsPage = lazy(() => import('./pages/ProductDetailsPage')

<Suspense fallback={<div>Загрузка...</div>}>
  <ProductDetailsPage />
</Suspense>
```

Рисунок 2.15. Пример применение Ract.lazy() и Suspense.

Такой подход особенно полезен для страниц, не используемых сразу (например личный кабинет или оформление заказа).

Lazy loading данных и изображений

Для повышения скорости загрузки страницы используется ленивая загрузка изображений и данных – они загружаются только тогда, когда реально необходимы (например, при прокрутке страницы).

Пример ленивой загрузки изображения вы можете видеть на рисунке 2.16.

```
<img src="placeholder.jpg" data-src="product.jpg" loading="lazy" />
```

Рисунок 2.16. Пример ленивой загрузки изображения.

Также можно использовать библиотеки, например, react-lazyload.

Мемоизация компонентов и значений

React рендерит компоненты каждый раз при изменении состояния. Чтобы избежать ненужных перерисовок, используются:

- React.memo() – для запоминания компонента;
- useMemo() – для запоминания вычисленных значений;
- useCallback() – для сохранения ссылок на функции.

Работа с большим количеством данных

При отображении большого количества товаров в каталоге или заказов в профиле важно избегать загрузки всей информации сразу. Используются:

- Пагинация – деление на страницы;
- Бесконечный скроллинг с динамической подгрузкой (infinite scroll);
- Отложенная загрузка карточек (по 10-20 элементов за раз).

Кэширование данных

Чтобы не загружать одни и те же данные при каждом переходе, можно использовать кэш:

- localStorage / sessionStorage – для хранения данных корзины, токена пользователя;
- RTK Query – встроенный механизм кэширования ответов;
- Библиотеки: SWR, React Query – для автоматического обновления и кеширования.

Минимизация ресурсов

При сборке проекта:

- Минифицируются файлы JavaScript и CSS;
- Удаляются неиспользуемые импорты (tree-shaking);
- Используются современные сборщики (например, Vite вместо Webpack для высокой скорости сборки);
- Включаются сжатие изображений и ресурсов (например, формат WebP вместо PNG/JPG).

Использования CDN

Статические ресурсы (шрифты, изображения, сторонние библиотеки) можно загружать с Content Delivery Network – это сокращает задержки и повышает скорость доставки.

Аудит производительности

Для оценки производительности приложения используется инструмент Lighthouse (встроен в Chrome DevTools). Он позволяет измерить:

- Время до первой отрисовки;
- Скорость загрузки;
- Отзывчивость интерфейса;
- Эффективность кэширования.

На основе отчёта можно принять решения о необходимых улучшениях.

Адаптация под мобильные устройства

Для повышения производительности на смартфонах:

- Используется адаптивная верстка;
- Скрываются/сворачиваются второстепенные элементы;
- Уменьшается объём анимаций и DOM-элементов.

Таким образом, оптимизация производительности – это совокупность мер, направленных на ускорение загрузки, снижение нагрузки и повышение отзывчивости интерфейса. Применение этих методов обеспечивает комфортную работу с приложением даже при медленном соединении и на маломощных устройствах, что особенно важно для интернет-магазинов с широкой аудиторией.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта была разработана одностраничная веб-система для онлайн-магазина с использованием технологии клиентского рендеринга (CSR) и библиотеки React. Проведённые теоретические исследования, архитектурное проектирование и реализация на практике подтвердили эффективность использования современных подходов к построению SPA для решений в сфере электронной коммерции.

В первой, теоретической части работы были рассмотрены ключевые понятия и технологии, лежащие в основе одностраничных приложений: особенности архитектуры SPA, принципы клиентской отрисовки, возможности React и его экосистемы. Особое внимание было уделено требованиям к интерфейсу онлайн-магазина и современным методам проектирования пользовательского опыта, включая адаптивность, модульной и интерактивность интерфейса.

Во второй, практической части, был последовательно реализован весь цикл разработки: от постановки задачи и формирования технических требований до реализации основных компонентом, маршрутизации, управления состоянием, взаимодействия с сервером и тестирования. В процессе разработки были использованы такие современные инструменты, как Redux Toolkit, React Router, Axios, а также применены подходы к оптимизации производительности: ленивые загрузки, разделение кода, мемоизация и кэширование.

Функциональность созданного приложения включает:

- Просмотр каталога и карточек товаров;
- Работа с корзиной;
- Оформление заказов;
- Регистрация и авторизация пользователей;
- Адаптивность интерфейса под различные устройства.

Результаты тестирования показали стабильную работу ключевых модулей, высокую отзывчивость интерфейса и корректную обработку пользовательских действий. Это подтверждает, что поставленная цель – создание SPA-приложения онлайн-магазина с применением CSR и React – была успешно достигнута.

Таким образом, разработанное приложение отвечает современным требованиям к веб-сервисам электронной торговли, отличается высокой производительностью, масштабируемостью и удобством использования. Оно может быть использовано как основа для реального интернет-магазина или расширено за счёт интеграции дополнительных функций: подключение платёжных систем, работа с внешними базами данных, SEO-оптимизация, мультиязычность и другие.

БИБЛИОГРАФИЯ

1. Сулыз А. В., Панфилов А. Н. Технология разработки одностраничного веб-приложения на платформе Angular 8 / Сулыз А. В., Панфилов А. Н. // *Молодой исследователь Дона*: [электронная версия]. – 2020. – № 2(23). – С. 69–72. – URL: https://cyberleninka.ru/article/n/tehnologiya-razrabotki-odnostranichnogo-veb-prilozheniya-na-platforme-angular-8 (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
2. Потовиченко М. А., Шатилов Ю. Ю. Разработка клиентской части одностраничного web-приложения с использованием библиотеки React / Потовиченко М. А., Шатилов Ю. Ю. // *Научное обозрение. Технические науки*: [электронная версия]. – 2020. – № 1. – С. 39–43. – URL: https://science-engineering.ru/ru/article/view?id=1278 (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
3. Газизуллин Н. И., Плещинская И. Е. Разработка клиентской части веб-приложения с использованием технологий SPA / Газизуллин Н. И., Плещинская И. Е. // *StudNet (научно-образоват. журнал)*: [электронная версия]. – 2020. – № 8. – С. 104–110. – URL: https://cyberleninka.ru/article/n/razrabotka-klientskoy-chasti-veb-prilozheniya-s-ispolzovaniem-tehnologiy-spa (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
4. Байдыбеков А. А., Гильванов Р. Г., Молодкин И. А. Современные фреймворки для разработки web-приложений / Байдыбеков А. А., Гильванов Р. Г., Молодкин И. А. // *Интеллектуальные технологии на транспорте*: [электронная версия]. – 2020. – № 2. – С. 15–21. – URL: https://cyberleninka.ru/article/n/sovremennye-freymvorki-dlya-razrabotki-web-prilozheniy (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
5. Байнов А. М., Кривоногова А. Е., Николаев А. С., Богомолова О. И. Обзор современных фреймворков и инструментов, используемых для разработки web-приложений / Байнов А. М., Кривоногова А. Е., Николаев А. С., Богомолова О. И. // *Наука без границ*: [электронная версия]. – 2020. – № 1(41). – С. 19–23. – URL: https://cyberleninka.ru/article/n/obzor-sovremennyh-freymvorkov-i-instrumetov-ispolzuemyh-dlya-razrabotki-web-prilozheniy (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.

6. Сергачева М. А., Михалевская К. А. Анализ фреймворков для разработки современных веб-приложений / Сергачева М. А., Михалевская К. А. // *Кронос: естественные и технические науки*: [электронная версия]. – 2020. – № 3. – С. 45–50. – URL: https://cyberleninka.ru/article/n/analiz-freymvorkov-dlya-razrabotki-sovremennyh-veb-prilozheniy (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
7. Бондаренко С. О. Современные интерактивные веб-приложения – построение пользовательского интерфейса с React / Бондаренко С. О. // *Вестник науки и образования*: [электронная версия]. – 2018. – № 5(41). – С. 46–48 (РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ОДНОСТРАНИЧНОГО WEB-ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ REACT - Научное обозрение. Технические науки (научный журнал)). – URL: https://cyberleninka.ru/article/n/sovremennye-interaktivnye-veb-prilozheniya-postroenie-polzovatelskogo-interfeysa-s-react (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
8. Сучков А. А., Гек Д. К., Багаева А. П. Использование ReactJS в современной web-разработке / Сучков А. А., Гек Д. К., Багаева А. П. // *Актуальные проблемы авиации и космонавтики*: [электронная версия]. – 2019. – № 2. – С. 378–380. – URL: https://cyberleninka.ru/article/n/ispolzovanie-reactjs-v-sovremennoy-web-razarabotke (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
9. Горбачев А. А., Горбачева Е. С. Сравнение классического процесса реализации веб-приложений и подхода с использованием библиотеки React / Горбачев А. А., Горбачева Е. С. // *Молодой исследователь Дона*: [электронная версия]. – 2020. – № 1(22). – С. 28–31. – URL: https://cyberleninka.ru/article/n/sravnenie-klassicheskogo-protsessa-realizatsii-veb-prilozheniy-i-podhoda-s-ispolzovaniem-biblioteki-react (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
10. Зиатдинов А. Р. Сравнение эффективности использования различных современных фронтенд-фреймворков в корпоративных приложениях / Зиатдинов А. Р. // *StudNet*: [электронная версия]. – 2020. – № 9. – С. 82–88. – URL: https://cyberleninka.ru/article/n/sravnenie-effektivnosti-ispolzovaniya-razlichnyh-sovremennyh-frontend-freymvorkov-v-korporativnyh-prilozheniyah (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
11. Мальцева В. В., Фролова А. С. Анализ web-технологий для создания онлайн-площадки / Мальцева В. В., Фролова А. С. // *Молодой исследователь*

Дона: [электронная версия]. – 2020. – № 2(23). – С. 126–130. – URL: https://mid-journal.ru/journal_content?id=2-2020 (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.

12. Кундубаев А. М., Вадиашвили Н. Н. Проектирование и разработка интернет-магазина компьютерной техники / Кундубаев А. М., Вадиашвили Н. Н. // *Теория и практика современной науки*: [электронная версия]. – 2018. – № 6(36). – С. 120–124. – URL: https://sciup.org/proektirovanie-i-razrabotka-internet-magazina-kompjuternoj-tehniki-140273655 (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
13. Володченко В. С., Ланцова Д. С., Метельницкая Т. А. и др. Актуальность и средства создания сайтов интернет-магазинов / Володченко В. С., Ланцова Д. С., Метельницкая Т. А., *и др.* // *Вопросы науки и образования*: [электронная версия]. – 2018. – № 2. – С. 15–21. – URL: https://cyberleninka.ru/article/n/aktualnost-i-sredstva-sozdaniya-saytov-internet-magazinov (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
14. Дрейзис Ю. И., Вершинина Г. Н. Анализ эффективности инструментария и средств автоматизации для электронной коммерции / Дрейзис Ю. И., Вершинина Г. Н. // *Вестник Академии знаний*: [электронная версия]. – 2019. – № 3. – С. 86–94 (ПЕРСПЕКТИВЫ РАЗВИТИЯ РЫНКА ИНТЕРНЕТ-ТОРГОВЛИ) ((PDF) Финансирование малых инновационных предприятий в ...). – URL: https://cyberleninka.ru/article/n/analiz-effektivnosti-instrumentariya-i-sredstv-avtomatizatsii-dlya-elektronnoy-kommertsii (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.
15. Галимзянов З. В. Разработка интернет-магазина / Галимзянов З. В. // *Научные междисциплинарные исследования*: [электронная версия]. – 2021. – № 2. – С. 55–59. – URL: https://cyberleninka.ru/article/n/razrabotka-internet-magazina (дата обращения: 22.05.2025). – Режим доступа: открытый доступ.