Постаногов Евгений Анатольевич

Бакалавр по специальности «Информационные системы и технологии»,

Псковский государственный университет, РФ, г. Псков

СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ ФОРМИРОВАНИЯ PDF-ДОКУМЕНТОВ В ВЕБ-ПРИЛОЖЕНИЯХ НА ОСНОВЕ REACT И NEXT.JS

Аннотация. В статье рассматриваются методы формирования PDF-документов в веб-приложениях на основе React и Next.js. Выполнено сравнение подходов, основанных на специализированных React-компонентах, декларативном описании документа, преобразовании HTML-страницы в PDF и растеризации содержимого. Определены критерии выбора метода с учётом точности отображения, производительности, редактирования и сопровождения шаблонов.

Abstract. The article examines methods for generating PDF documents in web applications based on React and Next.js. The study compares approaches based on specialized React components, declarative document description, HTML-to-PDF conversion, and rasterization of page content. The main selection criteria are defined with regard to visual accuracy, performance, document editing, and template maintenance.

Ключевые слова: react; next.js; pdf; html; веб-приложение; печатная форма; генерация документов; редактируемый документ.

Keywords: react; next.js; pdf; html; web application; print form; document generation; editable document.

Введение. Веб-приложения всё чаще применяются для подготовки документов, предназначенных для печати, отправки контрагентам, хранения в электронном архиве или передачи в другие информационные системы. К таким

документам относятся счета, акты, договоры, накладные, отчёты, паспорта качества, спецификации и другие формы, где важны сохранение структуры и одинаковый внешний вид при просмотре, печати и передаче получателю.

В приложениях на React и Next.js PDF-файлы могут создаваться как по отдельной структуре документа, так и на основе HTML-страницы [4; 8]. Выбор способа влияет на точность печатной формы, сложность разработки и нагрузку на систему.

Цель статьи – сравнить основные методы формирования PDF в веб-приложениях на основе React и Next.js. Рассматриваются решения на основе специализированных React-компонентов, декларативного описания, преобразования HTML-страницы через браузерный движок без графического интерфейса и растрового представления содержимого.

Постановка проблемы. В React- и Next.js-приложениях документ часто создаётся на основе данных, которые уже отображаются пользователю в веб-интерфейсе. Для типовых форм с постоянной структурой достаточно заранее описать расположение текста, таблиц, изображений и служебных реквизитов. Такой способ применим для простых отчётов, счетов, актов и иных документов, где внешний вид меняется незначительно.

Иная задача возникает при работе с документами, которые перед сохранением просматриваются и редактируются непосредственно в браузере. Пользователь изменяет отдельные значения, проверяет расположение элементов и ожидает, что итоговый PDF будет соответствовать последнему состоянию экранной версии. При повторном построении по отдельному шаблону возможны отличия в отступах, переносах строк, размере шрифта, ширине таблиц и расположении подписей.

Сопровождение усложняется, когда HTML-версия используется для просмотра и редактирования, а PDF создаётся по другой структуре. Каждое изменение шаблона требует синхронной правки обеих реализаций.

Основные подходы к формированию PDF-документов. В React- и Next.js-приложениях можно выделить четыре группы решений по исходному представлению документа.

Первый подход основан на использовании специализированных React-компонентов [1]. Разработчик описывает документ не через обычные HTML-элементы, а через компоненты, предназначенные для построения PDF. Такой способ близок к привычной модели React-разработки и подходит для документов с заранее определённой структурой.

Второй подход использует декларативное описание документа [5]. Структура задаётся в виде объекта, где указываются текстовые блоки, таблицы, изображения, стили и параметры расположения элементов. Этот вариант удобен для отчётов, счетов и документов с повторяемыми табличными данными, поскольку позволяет явно управлять составом и порядком блоков.

Третий подход предполагает преобразование HTML-страницы в PDF через браузерный движок без графического интерфейса [6; 7]. Документ сначала создаётся как обычная веб-страница с применением HTML и CSS, после чего сохраняется в печатный файл. Метод подходит для сценариев, где HTML-страница используется как основа печатного файла.

Четвёртый подход связан с переводом содержимого страницы в изображение [2; 3]. Отдельная область документа или весь блок интерфейса преобразуется в растровое представление, которое затем помещается в PDF-файл. Этот способ применим для простых визуальных выгрузок, но имеет ограничения при работе с текстом и многостраничными документами.

Критерии сравнения методов. Сравнение выполняется по шести критериям: точность отображения, поддержка сложной структуры, работа с редактируемым HTML-представлением, качество PDF-файла, производительность и сопровождение шаблонов.

Первым критерием является точность визуального отображения. PDF-документ должен сохранять заданную структуру страницы, размеры блоков, отступы, шрифты, таблицы и расположение графических элементов. Это важно

для форм, которые должны одинаково выглядеть при просмотре, печати и отправке внешним получателям.

Второй критерий связан с поддержкой сложной структуры документа. На практике печатная форма может содержать многострочные поля, вложенные таблицы, изображения, подписи, печати, служебные реквизиты и блоки с фиксированным положением на странице. Метод формирования PDF должен позволять управлять такими элементами без чрезмерного усложнения шаблона.

Третий критерий – возможность работы с редактируемым HTML-представлением. В этом случае PDF должен создаваться из того состояния документа, которое было показано и отредактировано в интерфейсе.

Четвёртым критерием является качество итогового PDF-файла. Текстовые данные должны оставаться доступными для поиска, выделения и копирования. При преобразовании содержимого в изображение эти возможности могут быть утрачены, а размер файла увеличивается.

Пятый критерий относится к производительности. Генерация PDF может выполняться на клиентской стороне, на сервере приложения или в отдельном сервисе. Выбранный подход определяет нагрузку на устройство пользователя, потребление ресурсов сервера и время формирования файла.

Шестой критерий связан с сопровождением шаблонов. Печатные формы часто изменяются: добавляются реквизиты, корректируются таблицы, меняются подписи, поля и правила отображения данных. Чем меньше дублирования между экранным представлением и механизмом создания PDF, тем проще поддерживать документ при развитии системы.

Сравнительный анализ методов. Сопоставление носит качественный характер и выполняется по критериям, обозначенным выше. Оценки отражают применимость каждого метода к типовым задачам подготовки печатных форм в React- и Next.js-приложениях: сохранение внешнего вида, поддержку сложной структуры, работу с редактируемым HTML-представлением, качество итогового файла, производительность и сложность сопровождения.

Таблица 1

Сравнение методов подготовки PDF-документов в React- и Next.js-приложениях

Критерий	Специализированные React-компоненты	Декларативное описание документа	Преобразование HTML-страницы в PDF	Растеризация содержимого страницы
Точность визуального отображения	Средняя	Средняя	Высокая	Нестабильная
Поддержка сложной структуры документа	Средняя	Высокая для таблиц	Высокая	Ограниченная
Работа с редактируемым HTML-представлением	Ограниченная	Ограниченная	Высокая	Частичная
Качество итогового PDF-файла	Текстовый слой сохраняется	Текстовый слой сохраняется	Текстовый слой сохраняется	Часто изображение вместо текста
Производительность	Средняя нагрузка	Средняя нагрузка	Повышенная нагрузка	Зависит от размера изображения
Сопровождение шаблонов	Отдельная PDF-разметка	Отдельная структура документа	Использование HTML-шаблона	Сложность растёт с объёмом
Примеры реализации	@react-pdf/renderer [1]	pdfmake [5]	Puppeteer, Playwright [6; 7]	html2canvas, jsPDF [2; 3]

Сравнение показывает различие между методами, которые строят PDF как самостоятельную структуру, и методами, использующими уже подготовленное экранное представление. Специализированные React-компоненты и декларативное описание дают разработчику контроль над содержанием документа, но требуют отдельной модели печатной формы. Это удобно для устойчивых шаблонов, отчётов и табличных документов, однако связь с пользовательским интерфейсом остаётся ограниченной.

Преобразование HTML-страницы в PDF занимает иную позицию: основой становится та же разметка, которая используется для просмотра или редактирования документа. Такой способ лучше подходит для форм, где важно сохранить внешний вид, проверенный пользователем в браузере. Недостатком является повышенная нагрузка на серверную часть, поскольку для создания файла используется браузерный движок.

Растеризация содержимого страницы имеет наиболее узкую область применения. Она позволяет быстро получить визуальную копию фрагмента интерфейса, но снижает качество работы с текстом. При таком способе PDF часто теряет полноценный текстовый слой, что ограничивает поиск, копирование и масштабирование. Для сложных многостраничных документов этот метод менее устойчив, чем решения, сохраняющие структуру текста и элементов.

Особенности реализации HTML-to-PDF-подхода в Next.js. В Next.js преобразование HTML-страницы в PDF может быть вынесено на серверную сторону [4; 6; 7]. Для этого в приложении создаётся отдельный маршрут, отвечающий за печатное представление документа. Такой маршрут получает данные из базы или из сохранённого состояния формы, формирует HTML-страницу с нужными стилями и передаёт её браузерному движку без графического интерфейса для последующего сохранения в PDF [6; 7].

При работе с редактируемыми документами сервер должен использовать не исходные данные, а последнее состояние формы, подтверждённое пользователем. Это может быть реализовано через сохранение изменённых значений перед генерацией PDF либо через передачу подготовленного набора данных в серверный обработчик. В обоих случаях печатный файл создаётся из той версии документа, которая была проверена в интерфейсе.

Для корректного результата печатная HTML-страница должна иметь отдельные правила отображения. В таких стилях задаются формат страницы, поля, размеры блоков, поведение таблиц, переносы строк, разрывы страниц и скрытие элементов, которые нужны только в интерфейсе. Особое внимание требуется к шрифтам, изображениям и внешним ресурсам: они должны быть доступны в момент создания PDF, иначе итоговый файл может отличаться от ожидаемой версии.

Серверная генерация через браузерный движок увеличивает потребление ресурсов [6; 7]. При единичном создании документов это ограничение обычно не критично, но при массовой подготовке файлов требуется контролировать число одновременных процессов. Для таких случаев генерацию целесообразно

выполнять через очередь задач или выносить в отдельный сервис, чтобы не перегружать основное приложение.

Рекомендации по выбору метода. Выбор способа подготовки PDF-документа должен начинаться с определения роли печатной формы в системе. Если документ создаётся по фиксированной структуре и не связан напрямую с экранным интерфейсом, можно использовать специализированные React-компоненты или декларативное описание. Для отчётов и документов с большим количеством строк предпочтительнее декларативный подход, поскольку он позволяет явно управлять таблицами, колонками и повторяемыми блоками.

Если документ просматривается или редактируется в браузере перед сохранением, приоритет получает преобразование HTML-страницы в PDF. Этот вариант снижает расхождения между проверенной пользователем версией и итоговым файлом. Он также подходит для печатных форм с фиксированной компоновкой, где важны шрифты, отступы, многострочные поля, изображения и расположение подписей.

Растеризацию содержимого страницы следует рассматривать как вспомогательный способ. Она допустима для простых визуальных выгрузок, но плохо подходит для документов, где требуется полноценный текстовый слой, поиск, копирование, небольшой размер файла и предсказуемое разбиение на страницы.

Отдельно необходимо учитывать место выполнения генерации. Клиентская подготовка PDF зависит от устройства пользователя и возможностей браузера. Серверная генерация даёт больший контроль над окружением, но требует управления ресурсами приложения. При массовой подготовке документов целесообразно использовать очередь задач или отдельный сервис генерации.

Заключение. Проведённый анализ показывает, что подготовка PDF-документов в React- и Next.js-приложениях должна рассматриваться как часть общей архитектуры работы с печатными формами. Значение имеет не только выбранная библиотека, но и источник данных, способ предварительного

просмотра, возможность редактирования, требования к структуре документа и условия выполнения генерации.

Ключевым фактором становится исходное представление документа. Если печатная форма описывается отдельно от интерфейса, система получает независимую модель PDF, но увеличивает объём сопровождения. Если основой выступает HTML-страница, сокращается число расхождений между проверяемой и сохраняемой версией, однако возрастает роль серверной обработки, настройки стилей печати и контроля ресурсов.

Практическая эффективность выбранного метода определяется не только качеством итогового файла. Не менее важны устойчивость шаблонов при изменении требований, предсказуемость результата, возможность массового формирования документов и сохранение корректного текстового слоя. Эти параметры позволяют оценивать генерацию PDF как технический процесс, связанный с развитием всей информационной системы.

Список литературы:

1. @react-pdf/renderer. Documentation // React-pdf: [сайт]. URL: https://react-pdf.org/ (дата обращения: 05.07.2026).

2. html2canvas. Documentation // html2canvas: [сайт]. URL: https://html2canvas.hertzen.com/documentation.html (дата обращения: 05.07.2026).

3. jsPDF. Documentation // jsPDF: [сайт]. URL: https://artskydj.github.io/jsPDF/docs/jsPDF.html (дата обращения: 05.07.2026).

4. Next.js. Documentation // Next.js : [сайт]. URL: https://nextjs.org/docs (дата обращения: 05.07.2026).

5. pdfmake. Documentation // pdfmake: [сайт]. URL: https://pdfmake.github.io/docs/0.1/ (дата обращения: 05.07.2026).

6. Playwright. Page // Playwright: [сайт]. URL: https://playwright.dev/docs/api/class-page (дата обращения: 05.07.2026).

7. Puppeteer. PDF generation // Puppeteer: [сайт]. URL: https://pptr.dev/guides/pdf-generation (дата обращения: 05.07.2026).

8. React. Documentation // React: [сайт]. URL: https://react.dev/ (дата обращения: 05.07.2026).