

В. А. Гилева^{1✉}, С. Ю. Кацко¹

Проектирование и реализация масштабируемого приложения для шаблонного создания структурированных отчетов

¹Сибирский государственный университет геосистем и технологий,
г. Новосибирск, Российская Федерация
e-mail: valeria.gileva.04@list.ru

Аннотация. Студенты зачастую затрачивают значительное время на самостоятельное форматирование учебных документов. Предоставление пользователю стандартного текстового редактора с избыточной функциональностью повышает риск возникновения ошибки оформления. В результате преподаватели возвращают подобные работы на доработку. В данной работе рассматривается опыт создания узкоспециализированного веб-инструмента. Логика работы системы построена на строгой регламентации: обучающийся осуществляет ввод содержательной части, а система обеспечивает автоматическое формирование итогового отчета. Техническая реализация проекта опирается на трехуровневую архитектуру на базе React и Node.js. Хранение разнородных фрагментов документа в классических реляционных таблицах представляется технически нецелесообразным. В качестве оптимального решения было выбрано применение формата JSONB в СУБД PostgreSQL, куда контент сохраняется в виде бинарного массива. Наиболее ресурсоемкие операции реализованы на серверной стороне. Взамен трудоемкой прямой реализации XML-разметки, разработанный алгоритм интегрирует данные веб-блоков в структуру базового эталонного шаблона DOCX. Изображения при этом масштабируются для обеспечения соответствия границам печатного листа. Таким образом, процессы форматирования автоматизируются. Снижается когнитивная нагрузка на студентов, а вероятность необходимости доработки из-за несоблюдения требований форматирования минимизируется.

Ключевые слова: генерация отчетов, веб-приложение, клиент-серверная архитектура, JSONB, автоматизация, шаблонизатор, React, Node.js

V. A. Gileva^{1✉}, S. Y. Katsko¹

Design and implementation of a scalable application for template-based creation of structured reports

¹Siberian State University of Geosystems and Technologies, Novosibirsk, Russian Federation
e-mail: valeria.gileva.04@list.ru

Abstract. Students often spend a considerable amount of time manually formatting academic documents. Providing users with a standard word processor featuring excessive functionality significantly increases the risk of formatting errors. As a result, instructors frequently return such assignments for revision. This paper discusses the practical experience of developing a highly specialized web-based tool. The system's operational logic is built on strict regulation: the student only inputs the core content, while the software ensures the automatic generation of the final report. The technical implementation of the project relies on a three-tier architecture based on React and Node.js. Storing heterogeneous document fragments within classic relational tables proved to be technically impractical. As an optimal workaround, the JSONB format was utilized within the PostgreSQL database, allowing all content to be saved as a single binary array. The most resource-intensive operations are strictly handled on the server side. Instead of laboriously generating XML markup from scratch, the developed algorithm seamlessly integrates web block data into the structure of a pre-configured DOCX reference

template. Uploaded images are programmatically scaled to ensure strict compliance with the borders of a standard printed page. Ultimately, the entire formatting process is fully automated. The cognitive load on students decreases significantly, and the probability of returning assignments due to formatting violations is reduced to a minimum.

Keywords: report generation, web application, client-server architecture, JSONB, automation, template engine, React, Node.js

Введение

Оформление документации по изучаемым дисциплинам требует существенных затрат учебного времени. Использование стандартных текстовых редакторов возлагает ответственность за нормоконтроль на самого студента. Избыточный функционал затрудняет концентрацию, что способствует увеличению количества неточностей. На поиск и исправление подобных технических недочетов затрачивается существенная часть времени проверки работ, что доказывает слабую эффективность ручного контроля [1].

Лидеры рынка вроде Microsoft Word визуально удобны. Однако слишком тесно связывают содержание документа с его представлением. Системы компьютерной верстки (LaTeX) исключают случайное изменение структуры. Данная степень надежности сопряжена с высокой сложностью освоения системы. Существуют также коммерческие блочные SaaS-платформы. PandaDoc удобен, но ориентирован исключительно на бизнес-контракты.

Выявляется потребность в разработке специализированного программного обеспечения. Цель исследования – проектирование и реализация масштабируемого приложения для шаблонного создания структурированных отчетов.

Разрабатываемое программное обеспечение призвано сочетать интерфейс блочного конструктора со строгим автоматическим контролем параметров форматирования. Это позволит студентам сфокусироваться на сути задания, а преподавателям – ускорить процесс проверки.

Методы и материалы

Проектирование системы базируется на классической трехуровневой клиент-серверной архитектуре. Разделение на визуальный интерфейс, вычислительный узел и подсистему хранения данных обеспечило необходимую гибкость.

Логика клиентской части написана на языке JavaScript с использованием библиотеки React. Выбор данной технологии обоснован ее компонентной архитектурой. За счет этой концепции массивную рабочую область удастся разделить на изолированные блоки (текст, изображение, список), пригодные для многократного встраивания без дублирования кода [2].

Схема взаимодействия компонентов интерфейса спроектирована с целью минимизации рисков совершения пользователем некорректных действий (рис. 1). Слева расположена панель инициализации команд, по центру – строго структурированная сетка редактора, а справа – изолированная колонка для ввода метаданных титульного листа.

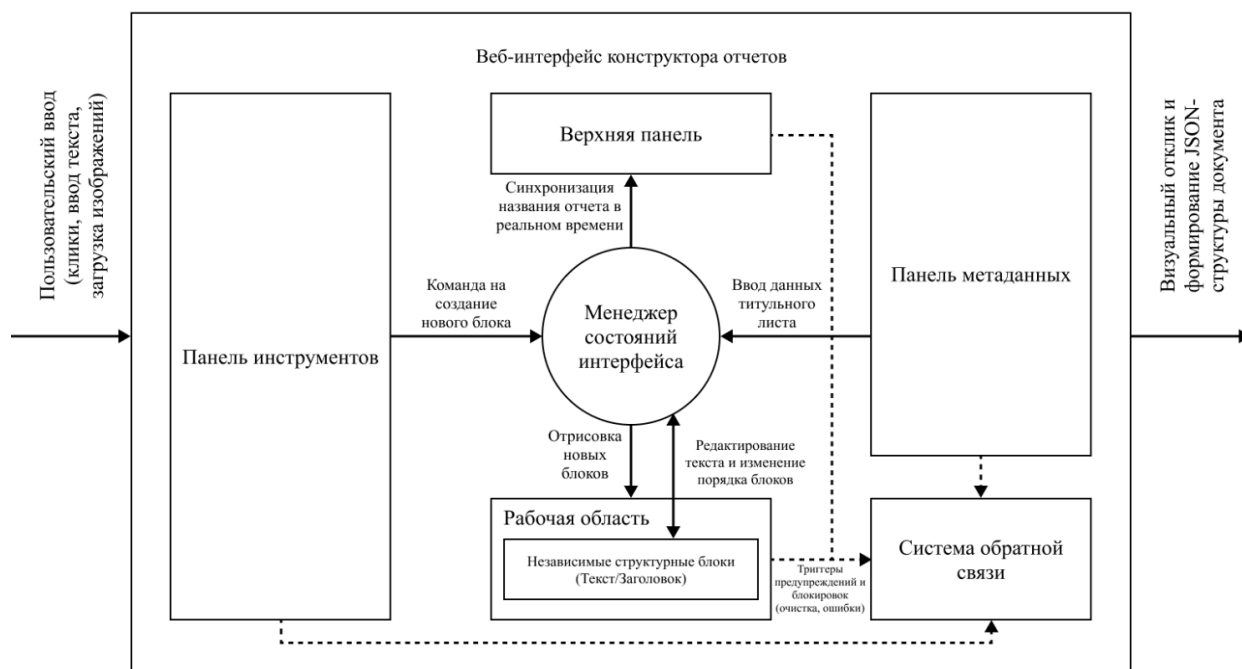


Рис. 1. Концептуальная модель взаимодействия компонентов интерфейса

Особое внимание при проектировании было уделено организации подсистемы хранения данных. Использование классической реляционной модели для хранения разноплановых элементов отчета (текста и графики) сопряжено с рядом технических ограничений. Решением стала разработка гибридной модели на базе объектно-реляционной СУБД PostgreSQL.

Метаданные организованы в типизированных столбцах. Слабоструктурированное содержимое документа хранится в отдельном поле типа JSONB (рис. 2). Динамический текст и изображения в кодировке Base64 предварительно преобразуются с помощью метода `JSON.stringify` для предотвращения ошибок сериализации глубоко вложенных структур, после чего сохраняются в базе данных в виде бинарных объектов.

reports

GeneralColumnsAdvancedConstraintsPartitionsParametersSecuritySQL

Inherited from table(s)Select to inherit from...

Columns

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	id	integer			☒	☒	nextval('r
	title	text			☒	☐	
	student_name	text			☐	☐	
	student_group	text			☐	☐	
	updated_at	timestamp without			☐	☐	CURREN
	meta_data	jsonb			☐	☐	
	content_data	jsonb			☐	☐	
	client_id	text			☐	☐	

Рис. 2. Структура хранения данных отчета в СУБД PostgreSQL

Серверный уровень реализован на платформе Node.js. Применение неблокирующей событийно-ориентированной модели ввода-вывода обеспечивает эффективную обработку параллельных асинхронных запросов [3]. Использование фреймворка Express позволило быстро настроить маршрутизацию. В целях предотвращения сбоев в работе сервера при загрузке объемных графических данных в кодировке Base64, базовый лимит пропускной способности был увеличен до 50 МБ.

Главная вычислительная задача сервера – сборка итогового файла. В качестве решения был реализован алгоритм заполнения эталонного шаблона.

В корневой директории сервера размещен файл-шаблон `template.docx`. Шаблон содержит предварительно заданные параметры форматирования, соответствующие требованиям ГОСТ. Для извлечения XML-структур из архива применяется утилита PizZip, а модуль `docxtemplater` используется для внедрения данных в шаблон, не затрагивая исходное форматирование полей и шрифтов [5]. За масштабирование графики отвечает библиотека `image-size`. Алгоритм считывает исходные размеры изображения. При превышении шириной изображения порогового значения в 600 пикселей, сервер вычисляет пропорциональный коэффициент и производит масштабирование, обеспечивающее соответствие графического объекта границам страницы формата А4 (рис. 3).



Рис. 3. Алгоритм серверной генерации документа

Сформированный бинарный буфер передается клиенту непосредственно. Реализованная архитектура не предусматривает запись промежуточных файлов формата DOCX на дисковый накопитель сервера.

Результаты

Базой для развертывания послужил арендованный виртуальный сервер под управлением Ubuntu. Функцию обратного проксирования взял на себя Nginx, тогда как фоновые процессы контролировались диспетчером PM2. По результатам проведенных испытаний заложенные в модули характеристики соответствуют заявленным требованиям.

Необходимо было подтвердить, что алгоритм фонового автосохранения не приведет к отказу базы данных вследствие избыточной частоты запросов. При эмуляции непрерывного ввода текста скорость поддерживалась на уровне,

превышающем 300 знаков в минуту. Избыточный сетевой трафик ограничивался на стороне клиента, а передача запросов к СУБД инициировалась только в тех случаях, когда в действиях пользователя фиксировалась пауза продолжительностью не менее двух секунд.

Техническое задание устанавливает предельно допустимое значение для одного пакета данных – 100 МБ. Попытка загрузить в редактор объемный графический архив размером 120 МБ привела к незамедлительному системному отказу. Веб-сервер Nginx заблокировал запрос на этапе приема, сформировав стандартную ошибку «413 Payload Too Large».

Оценка серверной компиляции файлов осуществлялась в условиях повышенной нагрузки. Эмулировалась одновременная активность пятидесяти пользователей. Размер входящих JSON-пакетов искусственно варьировался – изменялся объем графических данных, закодированных в Base64. Норматив генерации одного документа DOCX, согласно проектным требованиям, составляет 3-5 секунд. Анализ собранных статистических данных (табл. 1) свидетельствует о наличии значительного запаса производительности.

Таблица 1. Производительность серверной генерации отчетов

Сценарий наполнения	Прогнозируемый размер файла	Фактическое время обработки
Только текст	0,1 МБ	1,8 с
Текст и 3 изображения	~1,0 МБ	1,8 с
Текст и 30 изображений	8,0-9,0 МБ	2,0-3,0 с

График зависимости времени генерации от объема входных данных (рис. 4) имеет практически линейный характер. Отчет с большим объемом графических данных обрабатывается в установленные временные интервалы, а его итоговый размер значительно ниже критических значений.

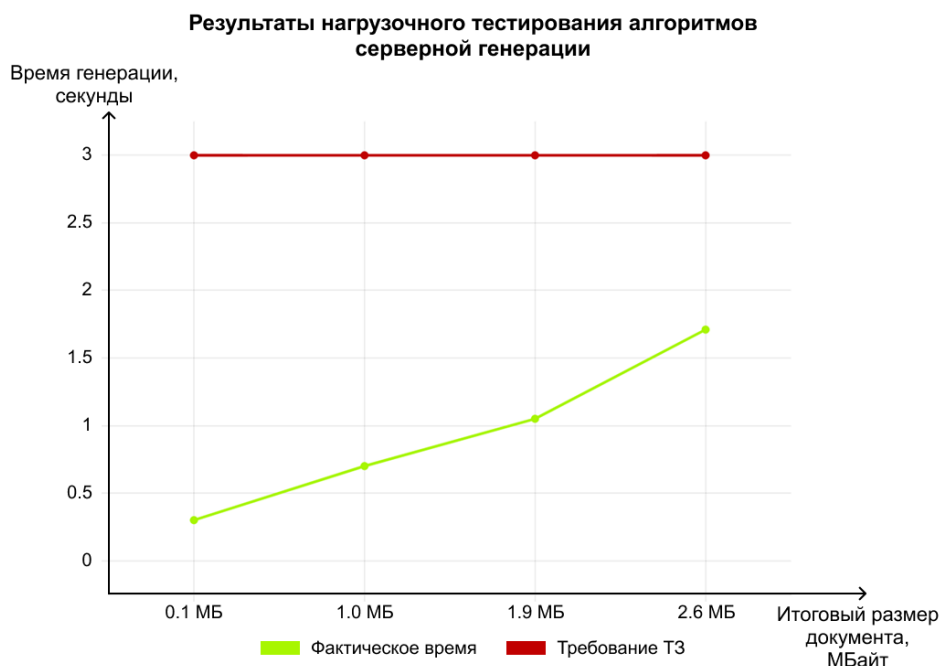


Рис. 4. График зависимости времени генерации документа от объема входных данных

Нарушений параметров форматирования в итоговых документах также не выявлено. Графические файлы большого объема масштабировались до ширины в 600 пикселей, вследствие чего выход за границы печатной области листа формата А4 во всех случаях был исключен.

В течение трех месяцев непрерывного функционирования системы средствами `pm2 monit` не было выявлено значимых утечек памяти (`Memory Leaks`). Оперативная память зафиксировалась на отметке 386 МБ. Полученные данные позволяют сделать вывод, что бинарные буферы `nodebuffer` удаляются из системы непосредственно в момент завершения скачивания готового архива клиентом.

Проверка кроссплатформенности проводилась в двух плоскостях. Во-первых, благодаря встроенному манифесту `Progressive Web App (PWA)`, редактор без проблем устанавливался прямо в среду `Windows`. Во-вторых, оценивалась сама `XML`-разметка готовых файлов. Проверка корректности отображения файлов осуществлялась посредством `Microsoft Word` и облачного сервиса `Google Документы`. Структура нигде не деформировалась, а визуально документы выглядели абсолютно идентично оригиналу.

Обсуждение

Апробация прототипа в условиях, приближенных к реальным, позволила выявить преимущества строгих архитектурных ограничений. Хранение данных с использованием формата `JSONB` в `PostgreSQL` позволило минимизировать количество избыточных связей между таблицами [4]. Базе данных не требуется

выполнение ресурсоемких транзакций при каждом добавлении пользователем нового блока в редактор.

Сравнение с классическим текстовым процессором Microsoft Word выявляет существенные различия в принципах работы. Нарушить верстку в данной системе не представляется возможным. В этом аспекте система близка к профессиональным средам, таким как LaTeX, которые также обеспечивают строгое соблюдение академических стандартов. Сложность освоения системы относительно невысока. Студентам не требуется изучать синтаксис разметки или устанавливать компиляторы. Удалось решить и типичную проблему коммерческих SaaS-конструкторов (таких как PandaDoc). При экспорте сохраняется исходная структура и форматирование документа. Строгая изоляция React-компонентов в браузере обеспечивает, что на сервер будет передана корректная иерархия блоков.

На стороне сервера алгоритм заполнения эталонного шаблона (модуль docxtemplater) продемонстрировал более высокую эффективность, чем внешние конвертеры, такие как Pandoc. Для архитектуры системы отсутствует необходимость инициализации нового процесса операционной системы для каждой генерации отчета. В отношении обработки графики конвертация изображений в Base64 стала обоснованным техническим компромиссом. Вследствие кодировки объем трафика возрастает приблизительно на треть, однако для формата MVP данное решение представляется оправданным.

Установленный уровень защиты является достаточным для автономного функционирования продукта. Доступ к сторонним отчетам ограничен посредством анонимных сессий, идентификаторы которых хранятся в локальном хранилище браузера. Однако данная модель обладает определенными техническими ограничениями, в частности, потенциальной уязвимостью к атакам межсайтового скриптинга (XSS). Поэтому перед возможным внедрением системы в информационную среду вуза необходимо будет провести дополнительное проектирование архитектуры безопасности.

Заключение

Требования технического задания реализованы в полном объеме. Результатом работы стал действующий масштабируемый веб-редактор, предназначенный для формирования структурированных отчетов на основе предварительно настроенного шаблона.

Выбранная трехуровневая архитектура продемонстрировала свою эффективность в ходе тестирования. Переход на формат данных JSONB внутри PostgreSQL обеспечил возможность структурирования динамического контента документа в виде единого массива.

На стороне клиента интерфейс (React) обеспечивает строгое соблюдение пользователем установленных ограничений. Для студентов минимизируется необходимость ручного контроля отступов и проверки соответствия иерархии шрифтов. Данные функции реализуются программным узлом на базе Node.js.

Алгоритм использует эталонный шаблон для адресного заполнения данными, при этом процесс формирования документа осуществляется в оперативной памяти.

Система продемонстрировала стабильность в ходе выполнения всех запланированных нагрузочных испытаний. Скорость работы соответствует установленным нормативам, поскольку время компиляции документов с большим объемом данных не превышает двух секунд. Обеспечение базового уровня информационной безопасности реализовано посредством механизма анонимных сессий с локальными ключами.

В перспективе одним из приоритетных направлений развития станет модернизация архитектуры информационной безопасности. Разработанный программный сервис может быть интегрирован в качестве функционального компонента в единую информационно-образовательную среду университета.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Стариченко, Б. Е. Программа автоматизации контроля оформления текстовых документов / Б. Е. Стариченко, М. А. Устинов. – Текст : электронный // Педагогическое образование в России. – 2018. – № 8. – С. 163–168. – URL: <https://cyberleninka.ru/article/n/programma-avtomatizatsii-kontrolya-oformleniya-tekstovyh-dokumentov> (дата обращения: 07.04.2026).
2. Мамбетов, Р. А. Построение пользовательских интерфейсов веб-приложений с помощью библиотеки React / Р. А. Мамбетов. – Текст : электронный // Экономика и социум. – 2019. – № 5 (60). – С. 906–909. – URL: <https://cyberleninka.ru/article/n/postroenie-polzovatelских-interfeysov-veb-prilozheniy-s-pomoschu-biblioteki-react> (дата обращения: 09.04.2026).
3. Стрелецкий, А. Д. Асинхронное программирование в Node.js: преимущества и подходы / А. Д. Стрелецкий, Н. М. Нетребин, С. Н. Русак. – Текст : электронный // Инновационная наука. – 2023. – № 11-2. – С. 43–44. – URL: <https://cyberleninka.ru/article/n/asinhronnoe-programmirovaniye-v-node-js-preimuschestva-i-podhody> (дата обращения: 09.04.2026).
4. Широков, И. А. Исследование производительности и методов хранения JSON-объектов в СУБД PostgreSQL / И. А. Широков. – Текст : электронный // Вестник НГУЭУ. – 2023. – № 3. – С. 144–153. – DOI: 10.34020/2073-6495-2023-3-144-153.
5. Колесинский, М. В. Автоматизация создания документов: совмещение шаблонов и диалогового взаимодействия / М. В. Колесинский, Н. И. Белодед. – Текст : электронный // Управление информационными ресурсами : материалы XX Международной научно-практической конференции. – Минск : Академия управления при Президенте Республики Беларусь, 2024. – С. 453–454. – URL: <https://elibrary.ru/item.asp?id=67323265> (дата обращения: 09.04.2026).

© В. А. Гилева, С. Ю. Кацко, 2026