МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет физико-технический**

**Кафедра теоретической физики и компьютерных технологий**

Допустить к защите

Заведующий кафедрой

д.ф.-м.н. наук, профессор

\_\_\_\_\_\_\_\_\_\_ К.А. Лебедев

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**(БАКАЛАВРСКАЯ РАБОТА)**

**РАЗРАБОТКА АЛГОРИТМА АВТОМАТИЗИРОВАННОЙ ПРОВЕРКИ ДОКУМЕНТАЦИИ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лукьянченко Николай Юрьевич

Направление подготовки 09.03.02 Информационные системы и технологии

Направленность (профиль) Программное обеспечение информационных систем

Научный руководитель

Профессор кафедры теоретической физики и компьютерных технологий, д-р физ.-мат. наук \_\_\_\_\_\_\_\_\_\_\_\_\_ Галуцкий Валерий Викторович

Нормоконтролер, преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ П. А. Василенко

Краснодар

2025

РЕФЕРАТ

Выпускная квалификационная работа 48 с., 0 рис., 0 табл., 10 источн.

DOCX, XML, WEB-САЙТ, TYPESCRIPT, CSS, TYPESCRIPT, NESTJS, NODE.JS

Предметом исследования является процесс разработки и внедрения веб-ориентированного сервиса для автоматизированного форматирования выпускных квалификационных работ в формате DOCX. В работе рассматриваются анализ требований к оформлению документов по стандартам вуза, проектирование клиент-серверной архитектуры, реализация модуля загрузки и обработки XML-контента, интеграция конфигурационного файла с правилами форматирования и оценка эффективности автоматизации рутинных операций.

Цель работы заключается в создании информационной системы, позволяющей автоматически приводить оформление DOCX-документов в соответствие с заданными требованиями (шрифты, отступы, выравнивание, стили заголовков и пр.), тем самым сокращая время и снижая число ошибок при подготовке ВКР.

Для достижения поставленной цели были сформулированы следующие задачи:

* изучить структуру формата DOCX и принципы его построения как ZIP-архива с внутренними XML-компонентами;
* проанализировать современные веб-технологии для разработки клиентской части (JavaScript, CSS) и серверной части (TypeScript, NestJS, Node.js);
* рассмотреть существующие библиотеки для парсинга и модификации XML в среде JavaScript/TypeScript;
* спроектировать REST-API для загрузки, обработки и доставки отформатированных документов;
* реализовать механизм распаковки/упаковки DOCX, чтения, изменения и сериализации XML через JSON-конфиг;
* обеспечить гибкость настроек форматирования через внешний JSON-файл и модульную архитектуру сервиса;
* провести тестирование корректности форматирования на реальных примерах и измерить производительность системы;
* организовать логику очистки временных файлов и мониторинг работы сервиса.

В результате исследования создан прототип веб-приложения, в котором с помощью JavaScript и CSS реализован интерфейс загрузки и скачивания документов, а серверная часть на TypeScript/NestJS обеспечивает автоматическую правку XML-структуры DOCX в соответствии с правилами из JSON-конфига. Система показала высокую скорость обработки (около 10–15 с. на документ массой 50 стр.) и значительно снизила число нарушений формата по сравнению с ручной правкой. Основные преимущества решения — полная автоматизация рутинных задач, гибкость настройки, кроссплатформенность и модульная архитектура, позволяющая расширять функционал (поддержка таблиц, списков, генерация оглавления и т. д.).

СОДЕРЖАНИЕ

[Введение 5](#_Toc198136780)

[1 Теоретические основы 8](#_Toc198136781)

[1.1 Формат документа docx 8](#_Toc198136782)

[1.2 Технологии веб-разработки 10](#_Toc198136783)

[1.3 TypeScript и NestJS 13](#_Toc198136784)

[1.4 Формат документа docx 16](#_Toc198136785)

[1.5 Стандарты оформления выпускных квалификационных работ 18](#_Toc198136786)

[2 Анализ и проектирование требований 22](#_Toc198136787)

[2.1 Сбор пользовательских требований 22](#_Toc198136788)

[2.2 Постановка задачи автоматизации форматирования 24](#_Toc198136789)

[2.3 Проектирование схем обработки документа 26](#_Toc198136790)

[3 Проектирование системы 28](#_Toc198136791)

[3.1 Общая архитектура решения 28](#_Toc198136792)

[3.2 Проектирование клиентского интерфейса 30](#_Toc198136793)

[4 Реализация практической части 32](#_Toc198136794)

[4.1 Разработка клиентской части 32](#_Toc198136795)

[4.2 Разработка серверной части 34](#_Toc198136796)

[4.3 Обработка и модификация xml 39](#_Toc198136797)

[4.4 Работа с файловой системой и архивами 41](#_Toc198136798)

[Заключение 45](#_Toc198136799)

[Список использованных источников 47](#_Toc198136800)

ВВЕДЕНИЕ

В последние годы наблюдается стремительный рост объёма научных и учебных работ, оформляемых в электронном виде. Формат DOCX стал де факто стандартом для подготовки выпускных квалификационных работ, дипломов, отчётов и статей. Вместе с тем растущие требования университетов и ГОСТ предъявляют строгие правила к оформлению документов: единообразные шрифты и их размеры, выравнивание текста по ширине, заданные отступы абзацев, оформление заголовков и разделов, корректная нумерация таблиц и рисунков. Ручная правка каждого документа под эти требования отнимает значительное время и часто приводит к ошибкам и несоответствиям.

Актуальность задачи. В условиях массового оформления и защиты ВКР студенты и преподаватели тратят много часов на исправление форматирования: выпускные квалификационные работы объёмом 50–100 страниц порой корректируются вручную по несколько раз. Автоматизация этого процесса позволяет существенно сократить время подготовки документов, минимизировать вероятность ошибок и стандартизировать оформление по единому шаблону. Это особенно важно при большом числе студентов, когда нагрузка на службу приёма и кафедры возрастает, а качество форматирования напрямую влияет на оценку работы.

Цель работы заключается в создании веб-ориентированной системы для автоматизированного форматирования документов DOCX в соответствии с заранее заданными требованиями (шрифты, размеры текста, отступы, выравнивание, стили заголовков и разделов и т. д.).

Для достижения поставленной цели необходимо решить следующие задачи:

1. Исследовать внутреннюю структуру формата DOCX как ZIP-архива с компонентами XML и определить ключевые файлы, влияющие на визуальное оформление документа.
2. Проанализировать существующие подходы и библиотеки для парсинга и модификации XML в среде JavaScript/TypeScript (xml2js, fast-xml-parser и др.).
3. Спроектировать архитектуру клиент-серверного веб-приложения, включающую интерфейс загрузки и скачивания документов, REST-API и модуль обработки XML.
4. Разработать клиентскую часть на JavaScript и CSS для удобной работы пользователя с загрузкой исходного файла и получением результата.
5. Реализовать серверную часть на TypeScript с использованием фреймворка NestJS, обеспечивающую распаковку DOCX, применение правил форматирования из JSON-конфига и упаковку обновлённого документа.
6. Организовать гибкую систему конфигурации правил форматирования: создание и валидация внешнего JSON-файла с параметрами (шрифты, размеры, отступы, стили заголовков).
7. Провести модульное, интеграционное и нагрузочное тестирование сервиса: оценить корректность форматирования и производительность обработки различных по объёму документов.
8. Оценить практическую эффективность автоматизации: сравнить время и количество ошибок при автоматическом и ручном форматировании на выборке реальных ВКР.

Объектом исследования является процесс автоматизированного форматирования документов в формате DOCX, включающий операции разбора, модификации и сборки XML-компонентов внутри ZIP-архива.

Предметом исследования выступает разработка и внедрение веб-приложения для автоматической правки оформления VКР, а также оценка его эффективности при использовании в университете.

Теоретическая значимость работы заключается в обобщении и анализе методов программной работы с DOCX-документами, исследовании возможностей XML-парсинга в среде JavaScript/TypeScript и применении архитектурных подходов NestJS для решения задач автоматизации форматирования.

Практическая значимость исследования заключается в создании прототипа веб-сервиса, который:

* сокращает время подготовки выпускных работ от 2–3 часов ручной корректировки до 10–15 секунд автоматической обработки;
* обеспечивает единообразие оформления по заданным стандартам;
* минимизирует человеческие ошибки при форматировании;
* легко расширяется и настраивается через внешний JSON-конфиг.

Результаты проведённого исследования и тестирования прототипа показали высокую производительность и надёжность системы. Дальнейшее развитие проекта может быть направлено на поддержку сложных элементов DOCX (таблицы, списки, оглавление) и интеграцию с образовательными порталами и MS Word-надстройками.

# **Теоретические основы**

## **Формат документа docx**

Формат DOCX представляет собой современный стандарт текстовых документов, введённый корпорацией Microsoft в составе пакета Microsoft Office 2007 и ставший преемником устаревшего бинарного формата DOC. Основной задачей перехода на DOCX стало использование открытой спецификации Office Open XML (OOXML), подготовленной консорциумом ECMA и позже одобренной ISO/IEC. Преимущества нового формата заключаются в его модульности, высокой степени сжатия и открытости спецификации для сторонних разработчиков.

DOCX-файл по своей сути является ZIP-архивом, внутри которого хранится набор XML-документов и вспомогательных ресурсов (изображений, шрифтов, стилей). Такая структура позволяет эффективно сжимать данные, поскольку XML-текст хорошо поддаётся алгоритмам ZIP, а также упрощает программную обработку: любой инструмент, умеющий читать ZIP и парсить XML, может анализировать и модифицировать содержимое документа.

Основные компоненты DOCX-файла:

* [Content\_Types].xml – файл, описывающий типы контента и связи с внутренними частями пакета.
* \_rels/.rels – глобальные связи между компонентами пакета и внешними ресурсами.
* word/document.xml – центральный файл, содержащий основное содержимое документа: текст, абзацы, разделы.
* word/styles.xml – определения стилей абзацев и символов (шрифтовый набор, размеры, форматирование).
* word/\_rels/document.xml.rels – связи документа с другими ресурсами (такими как изображения, таблицы стилей).
* word/media/ – директория с медиа-файлами: рисунки, графика, встроенные объекты.
* word/theme/, word/fontTable.xml и др. – дополнительные файлы, отвечающие за оформление, темы и таблицы шрифтов.

Внутри document.xml текст представляется через последовательности XML-элементов:

* <w:p> – параграф, внутри которого
* <w:r> – «бегунок» (run) – отрезок текста с едиными свойствами, содержащий
* <w:t> – собственно текстовую строку.

Атрибуты элементов задают такие параметры, как выравнивание (<w:jc w:val="both"/> для выравнивания по ширине), отступы (<w:ind w:firstLine="720"/> для отступа первой строки) и интервал между строками (<w:spacing w:line="360"/>). Элементы <w:pPr> и <w:rPr> отвечают за свойства параграфа и «бегунка» соответственно.

Одной из ключевых особенностей формата является использование ссылок внутри ZIP-архива: вместо дублирования больших двоичных объектов (например, изображений) они хранятся один раз в word/media/, а в XML-ссылках указывается относительный путь через файл .rels. Такой подход снижает размер итогового документа и упрощает управление связями.

Многие сторонние библиотеки для работы с DOCX (например, docx4j на Java, python-docx на Python, а в экосистеме JavaScript — mammoth.js, docxtemplater, officegen) используют комбинацию ZIP-распаковки и XML-парсинга. Общий алгоритм обработки DOCX выглядит так:

1. Распаковка ZIP-контейнера на файловую систему или виртуальный файловый объект.
2. Чтение и парсинг word/document.xml и смежных файлов через XML-парсер.
3. Модификация узлов DOM-дерева в соответствии с требованиями (например, изменение w:val атрибутов у <w:rPr> для выставления единого шрифта и размера текста).
4. Сериализация изменённого XML обратно в текст.
5. Запаковка всех файлов в новый ZIP-архив с расширением .docx.

Спецификация OOXML подробно описывает структуру каждого файла и назначение элементов, что позволяет точечно вмешиваться в документ без риска повредить его целостность. Однако необходимо учитывать нюансы: разные версии Microsoft Word могут по-разному интерпретировать некоторые атрибуты или добавлять собственные расширения XML-namespace. Поэтому при разработке автоматизированных средств форматирования важно протестировать работу с документами, созданными в разных офисных пакетах и версиях.

Таким образом, понимание внутренней структуры DOCX и принципов работы с ZIP-архивами и XML-документами лежит в основе автоматизации форматирования ВКР: знание расположения ключевых файлов, структуры узлов и атрибутов позволяет программно корректировать шрифты, отступы, выравнивание и другие параметры оформления без ручного вмешательства.

## **Технологии веб-разработки**

Развитие веб-технологий за последние десятилетия превратило браузер из простого инструмента чтения статических страниц в полноценную платформу для интерактивных приложений. На стороне клиента основным языком остаётся JavaScript, который из скриптового «последнего довеска» превратился в мощный инструмент для управления логикой и состоянием интерфейса. В самом начале страницы загружается минимальный HTML-каркас, а уже с помощью JavaScript происходит динамическая подстановка элементов, их стилизация и реакция на действия пользователя. Когда посетитель нажимает кнопку «Загрузить файл», начинается чтение содержимого DOCX-документа средствами FileReader, а затем формируется объект FormData и через современный API fetch отправляется на сервер. Всё это происходит без перезагрузки страницы, позволяя добиться плавного и дружелюбного взаимодействия.

При этом управление структурой и содержимым страницы основано на понятии DOM – дерева узлов, где каждый элемент HTML превращается в объект с возможностью читать и изменять его свойства в реальном времени. С помощью методов querySelector и getElementById скрипт находит нужный контейнер для формы, затем присоединяет к нему элементы: поля ввода, прогресс-бар и сообщения об ошибках. При изменении размеров окна браузера JavaScript слушает события resize и меняет расположение элементов, а при наведении курсора над интерактивными зонами запускает анимации, делая интерфейс живым и отзывчивым.

Однако красота и удобство интерфейса во многом зависят от того, как стилизован HTML. Здесь на помощь приходит CSS – каскадные таблицы стилей, где каждый селектор, правило и псевдокласс отвечают за цветовую гамму, шрифты и позиционирование блоков. Строя макет страницы, разработчик опирается на модель коробок (box model), где каждый элемент состоит из содержимого, внутренних полей, рамок и внешних отступов. Именно правильное управление этими отступами делает форму для загрузки файла аккуратной и гармонирующей с общим дизайном. При создании прогресс-бара дизайнеры часто не прибегают к сложному JavaScript, а используют псевдоэлементы и ключевые кадры CSS, изменяя ширину заполненного сегмента и добиваясь плавного эффекта заполнения, лишь программно обновляя класс или значение CSS-свойства.

С переходом к более сложным макетам без возможностей современных CSS-модулей было трудно создавать адаптивные страницы. Flexbox решил эту проблему для одномерных рядов, предоставив контейнеру режим дисплея «гибкой коробки», где дочерние элементы автоматически распределяются вдоль основной или поперечной оси, равномерно занимая доступное пространство. Если ширина экрана уменьшается, Flexbox позволяет изменить направление выравнивания, переведя строки в столбцы, и расположить кнопки друг под другом, не меняя при этом саму структуру HTML.

Тем не менее Flexbox облегчает работу лишь в одном измерении. Для комплексных макетов, когда нужно заполнить пространство сеткой, используется CSS Grid. Этот двухмерный инструмент позволяет разбить страницу на зоны, подобно клеткам таблицы, и поместить в каждую из них свои компоненты. Например, навигационная панель, основная область работы и футер могут быть описаны зонами одном шаблоном, после чего элементы размещаются с помощью указания номеров строк и столбцов. При изменении размеров экрана Grid пересчитывает размеры ячеек, сохраняя пропорцию и адаптируя дизайн под новое разрешение.

Говоря об адаптивности, нельзя обойти внимание методику «mobile first», когда стили сначала пишутся для узких экранов и только потом расширяются медиазапросами для более широких устройств. В нашей системе пользователь может открыть страницу на смартфоне: строки скрипта для работы с FileReader, форма загрузки и индикатор прогресса распределяются по вертикали и занимают всю ширину экрана, а шрифт автоматически уменьшается для удобного чтения. По мере расширения окна добавляются дополнительные отступы и плавные переходы между колонками, обеспечивая комфортную работу и на планшете, и на мониторе ноутбука.

Именно благодаря совокупности JavaScript, CSS, Flexbox, Grid и адаптивных приёмов удалось создать простой, но при этом мощный интерфейс, который обеспечивает пользователю возможность загрузить свой документ, следить за ходом обработки и получить результат без перерывов и лишних кликов. Эта связка технологий лежит в основе клиентской части проекта и гарантирует, что сервис будет одинаково удобен как для студентов, так и для преподавателей, независимо от используемого устройства и браузера.

## **TypeScript и NestJS**

TypeScript и NestJS стали ключевыми технологиями в разработке серверной части проекта благодаря своей строгости, масштабируемости и удобству поддержки. В отличие от традиционного JavaScript, где типизация носит динамический характер, TypeScript добавляет статическую типизацию, что особенно важно в крупных проектах. Благодаря этому уже на этапе компиляции можно выявить целый ряд ошибок: несоответствие типов, обращение к несуществующим свойствам, неправильная структура объекта. Это не только ускоряет отладку, но и делает код предсказуемым и более безопасным. В проекте, где обрабатываются файлы формата DOCX, такая строгость критична, ведь данные поступают от пользователя, а значит, имеют потенциально непредсказуемую структуру. Проверка типов позволяет избежать падения приложения и упрощает обработку исключений.

TypeScript также облегчает работу с современными возможностями JavaScript, добавляя к ним аннотации типов, интерфейсы и дженерики. Это делает API-функции более читаемыми и удобными в использовании. Например, при создании сервиса, который обрабатывает содержимое XML-файлов, удобно определить интерфейс, описывающий структуру параграфа, его свойства и допустимые стили. Это помогает избежать неявных ошибок и облегчает дальнейшее масштабирование проекта. Код становится самодокументируемым: один взгляд на сигнатуру функции достаточно, чтобы понять, какие данные она принимает и что возвращает.

Для архитектуры сервера был выбран NestJS — фреймворк, построенный поверх Node.js и использующий TypeScript изначально. NestJS перенимает лучшие идеи из мира объектно-ориентированного программирования и фреймворков, таких как Angular и Spring. Он предлагает чёткую модульную структуру, поддержку инъекции зависимостей, декораторы и встроенные механизмы маршрутизации. Это упрощает разделение приложения на логические слои: контроллеры, сервисы, модули и посредники. Такая архитектура позволяет удобно масштабировать проект: добавить новый функционал — например, загрузку нескольких документов — можно, не затрагивая остальную часть кода.

Важной особенностью NestJS является его строгая организация проекта. Каждый модуль отвечает за конкретную область ответственности: загрузка файлов, проверка форматирования, управление конфигурациями, логгирование ошибок. Это снижает связанность компонентов и упрощает тестирование. Модульность позволяет не только изолировать код, но и повторно использовать его. Например, логика, отвечающая за проверку стилей заголовков и текста, реализована в виде отдельного сервиса и может быть вызвана из разных маршрутов или использоваться в очередях задач для асинхронной обработки.

NestJS также предлагает встроенную поддержку асинхронности через async/await и Promises, что критично в задачах работы с файлами, сетью или базой данных. Обработка загруженного DOCX-файла начинается с получения бинарных данных, затем происходит его распаковка, парсинг XML и проверка содержимого. Все эти операции выполняются без блокировки потока выполнения, что позволяет обрабатывать несколько файлов одновременно и обеспечивать стабильную работу сервера даже при высокой нагрузке.

Для удобства конфигурации в NestJS используется встроенный модуль ConfigModule, который позволяет загружать переменные окружения из .env-файлов. Это обеспечивает удобную адаптацию приложения под разные среды: локальную, тестовую и продакшн. Благодаря этому легко менять пути к временным папкам, ограничения по размеру файла или включать логгирование в зависимости от режима запуска.

Неотъемлемой частью NestJS является встроенный модуль маршрутизации. С его помощью были созданы REST-эндпоинты, обрабатывающие загрузку и валидацию файлов. Каждый контроллер чётко описывает свою зону ответственности. Например, контроллер, принимающий POST-запрос с DOCX-документом, делегирует проверку содержимого валидационному сервису, а затем возвращает пользователю отчёт о несоответствиях. Использование декораторов @Post, @Body, @UploadedFile и других позволяет легко читать код и быстро находить нужные маршруты.

Особое внимание в проекте уделено логике обработки формата DOCX. Этот формат, по сути, является архивом ZIP, содержащим XML-файлы с форматированием, стилями, текстами и другими метаданными. С помощью библиотек, работающих на стороне сервера, содержимое архива извлекается, далее XML-файл document.xml анализируется и проходит проверку на соответствие определённым правилам. Благодаря TypeScript, описание структуры XML-документа стало более удобным, а навигация по его содержимому — безопасной и контролируемой.

Таким образом, связка TypeScript и NestJS обеспечивает проекту надёжность, читаемость, удобство поддержки и масштабируемость. Это особенно важно в дипломной работе, где требуется не только реализовать функциональность, но и соблюсти стандарты проектирования, обеспечения безопасности и расширяемости. Такой подход демонстрирует серьёзный профессиональный уровень в области разработки серверных решений.

## **Формат документа docx**

Формат DOCX представляет собой один из самых распространённых форматов электронных документов, используемый в современном документообороте. Он был официально представлен компанией Microsoft в 2007 году вместе с выходом пакета Office 2007, где стал преемником формата DOC, существовавшего с начала 1990-х годов. Главное отличие нового формата заключается в его структуре: если DOC представлял собой бинарный файл, то DOCX основан на стандартах открытой XML-структуры и фактически является ZIP-архивом, содержащим набор взаимосвязанных XML-документов и вспомогательных файлов. Такая архитектура обеспечила гибкость, расширяемость и стандартизацию, сделав DOCX не только удобным для пользователя, но и доступным для программной обработки.

Каждый DOCX-документ можно открыть и просмотреть как обычный архив, если изменить его расширение на .zip. Внутри архива располагаются файлы и папки, в которых хранится информация о содержимом документа, его структуре, стилях, форматировании, встроенных объектах и даже настройках. Центральным элементом является файл word/document.xml, содержащий основной текст документа с разметкой, описывающей абзацы, шрифты, стили, отступы, выравнивание, вложенные элементы и многое другое. Эта структура представляет собой XML-документ, в котором каждый текстовый элемент помещён в тег <w:t>, а абзацы — в <w:p>, что позволяет точно определить, где начинается и заканчивается конкретная текстовая единица.

Одним из преимуществ формата DOCX является его модульность: каждая составляющая документа — будь то таблицы стилей (styles.xml), настройки шрифта (fontTable.xml), сноски (footnotes.xml), изображения (media) или структура содержания (numbering.xml, settings.xml) — располагается в отдельном файле. Это обеспечивает удобство при обработке документов с использованием программных средств. Например, для анализа правильности форматирования текста необходимо распаковать документ, затем с помощью парсера XML прочитать document.xml и проверить каждый текстовый элемент и его родительские теги на соответствие требованиям — будь то выравнивание по центру, наличие жирного начертания, правильный шрифт или размер текста.

Поскольку DOCX является открытым стандартом (частью спецификации Office Open XML, принятой ECMA и ISO), он может быть использован не только в продуктах Microsoft, но и в сторонних редакторах и приложениях. Это делает его универсальным форматом для обмена документами, что особенно важно при разработке автоматизированных систем, работающих с документами. Во многих языках программирования, включая JavaScript и TypeScript, доступны библиотеки, позволяющие взаимодействовать с DOCX-файлами: читать их, изменять, создавать отчёты, проверять содержимое и экспортировать результаты. На серверной стороне такие библиотеки часто используют для валидации, генерации шаблонов, создания динамических отчётов и конвертации.

В контексте дипломной работы формат DOCX выбран не случайно. Он представляет собой стандарт, активно используемый в образовательных учреждениях, государственных организациях и бизнес-среде. Для проверки оформления таких документов на соответствие методическим требованиям (например, шрифт Times New Roman, размер 14, выравнивание по ширине, определённые отступы и междустрочный интервал) необходима точная и надёжная технология анализа. Используя формат DOCX, можно извлекать структуру документа, обходить элементы XML, определять параметры каждого абзаца и проверять, соответствуют ли они утверждённым правилам. Это возможно благодаря понятной и документированной структуре формата, а также наличию метаданных, описывающих стили и параметры текста.

Также важно отметить, что работа с DOCX требует внимания к пространствам имён XML. Так как все теги и атрибуты в файле содержат префиксы (например, <w:t>, <w:pPr>), необходимо учитывать их при парсинге и обращаться к тегам с учётом пространства имён w, которое ссылается на URI из спецификации Microsoft. Игнорирование этих нюансов может привести к невозможности корректной обработки структуры документа и ошибкам при анализе.

Таким образом, DOCX — это не просто формат хранения текста, а целая система, описывающая документ с точки зрения структуры, внешнего вида и поведения. Его использование в рамках дипломного проекта предоставляет гибкий и мощный инструмент для автоматического анализа документов, выявления несоответствий требованиям и генерации отчётов. Именно благодаря своей доступности, стандарту и архитектуре формат DOCX становится незаменимым в системах, предназначенных для анализа и контроля качества документов.

## **Стандарты оформления выпускных квалификационных работ**

Текст работы оформляется в соответствии с ГОСТ 7.32 и ГОСТ 2.105, при этом для набора рекомендуется использовать тот текстовый редактор, который принят в вашей образовательной организации. Выпускную квалификационную работу и курсовые проекты печатают на белой односортной бумаге формата А4 односторонним способом через полтора интервала. Поля задаются так: слева 30 мм, справа 15 мм, сверху и снизу по 20 мм. Основной текст набирают шрифтом Times New Roman 14 пт чёрным цветом, выравнивая по ширине и допуская автоматические переносы. Абзацный отступ устанавливают 1,25 см. При необходимости выделить отдельные термины или формулы можно применять другие гарнитуры шрифта, сохраняя общую плотность текста.

Объём ВКР должен быть не менее 50 страниц машинописного текста (без приложений), при этом обзорная часть может занимать до трети всего объёма. Курсовые проекты составляют 15–20 страниц на втором курсе и 20–30 страниц на третьем. Работы подшивают в твёрдую папку или используют скоросшиватель; кольцевые папки и файлы-вкладыши не рекомендуются. Вписывать в текст отдельные слова, формулы или условные обозначения следует чёрными чернилами по ГОСТ 2.304, чтобы плотность вписанного материала совпадала с печатным текстом. Опечатки и рисунки, в которых обнаружены ошибки, разрешается исправлять подчисткой или закрашиванием белой краской, а затем вносить исправления машинописным способом или чёрными чернилами; если исправлений более четырёх на странице, её целиком перепечатывают.

Пустых или полупустых строчек и страниц быть не должно. Нумерация выполняется арабскими цифрами: титульный лист учитывается в общем счёте, но без номера, страницы со второго номера ставят внизу по центру. Для содержания наименования подразделов и пунктов отступают на два знака от разделов и подразделов, соединяя название с номером при помощи точек. Заголовки таких элементов, как «Реферат», «Содержание», «Введение», «Заключение», «Список литературы» и «Приложения», печатают прописными буквами по центру с новой страницы и отделяют от текста одной строкой. Заголовки разделов и подразделов набирают строчными буквами с первой прописной, выделяют полужирным, но без подчёркивания и точки в конце, при этом каждый раздел начинается с новой страницы. Нумерация пунктов и подпунктов идёт арабскими цифрами через точку без точки в конце, например 2.1 или 2.2.1. Переносы слов в заголовках недопустимы.

Перечисления внутри пунктов оформляют без маркеров: простые — через запятую, сложные — через точку с запятой. При ссылке на элементы списков вместо тире используют строчные буквы русского алфавита (а, б, в и т. д.), а для более глубокой детализации — арабские цифры в скобках. По возможности строки заголовков структурных элементов набирают средствами многоуровневых стилей («Заголовок 1», «Заголовок 2» и т. д.), корректируя их параметры.

Примечания размещают сразу после относящегося к ним текста или иллюстрации с абзацным отступом и одной свободной строкой сверху. Если в работе одно примечание, его не нумеруют: слово «Примечание» пишут с прописной, затем ставят тире и текст. При нескольких примечаниях используют арабские цифры без точки.

Иллюстрации располагают непосредственно после первого упоминания в тексте или на следующей странице; допускается несколько рисунков на одном листе. Все рисунки нумеруются сквозной арабской нумерацией или в пределах раздела с обозначением «1.1», «1.2» и т. д. Слово «рисунок» в тексте пишут строчными, в подписи — с прописной, после номера ставят тире. Листы с изображениями, выполненными на формате А3 и сложенными до А4, учитываются как одна страница. Длинные названия рисунков набирают в одинарный интервал без переносов слов. Иллюстрации приложений маркируются отдельно с добавлением буквы приложения, например «Рисунок A.3».

Таблицы оформляют так: заголовок располагается над таблицей слева в одну строку с её номером через тире, шрифт и межстрочный интервал могут быть меньше, чем в тексте. Таблицы нумеруются сквозной арабской нумерацией или в пределах раздела. Если таблицу продолжают на следующей странице, над продолжением указывают «Продолжение таблицы X» и повторяют заголовки столбцов. Таблицы располагают сразу после первого упоминания.

Формулы выделяют отдельной строкой, нумеруя арабскими цифрами в круглых скобках справа, оставляя по одной свободной строке до и после. Пояснения к символам и коэффициентам приводят под формулой, начиная слово «где» без двоеточия. При переносах формул строку переносят после знака равенства или основных операторов, повторяя знак.

Единицы физических величин и их обозначения оформляют по ГОСТ 8.417: используют международные или русские обозначения (одновременно оба не применяют), между числом и единицей ставят неразрывный пробел, предельные отклонения указывают скобками или знаком «±», а интервалы записывают словом «от»…«до». Числа с единицами пишут цифрами, числа от одного до девяти без единиц — словами. Для произведений единиц применяют точку на средней линии, для дробей — одну черту.

Ссылки на литературу в тексте оформляют порядковым номером в квадратных скобках и приводят в списке в порядке первого упоминания. При ссылках на формулы используют круглые скобки, на рисунки и таблицы — словосочетания «рисунок X» или «таблица Y». Список литературы печатают тем же шрифтом через полтора интервала, нумеруя арабскими цифрами с точкой и не допускают разрыва элементов описания при переносах строк.

**Приложения оформляют продолжением работы: каждое начинается с новой страницы с заголовком «ПРИЛОЖЕНИЕ» и его буквенным обозначением, набирают полужирным прописными по центру. Приложения маркируют заглавными буквами (А, Б, В и т. д.), все страницы приложений включают в общую нумерацию, а ссылки на них дают в тексте.**

# **Анализ и проектирование требований**

## **Сбор пользовательских требований**

Приступая к разработке системы автоматизированного форматирования DOCX-документов, мы начали с глубокого изучения того, какие именно потребности испытывают конечные пользователи – студенты, научные руководители и администраторы приёма работ. В первую очередь нам нужно было выяснить, какие этапы подготовки выпускной квалификационной работы создают наибольшие трудности, какие действия повторяются вручную и сколько времени уходит на каждую из них. Для этого были проведены личные интервью со студентами разных факультетов, которые уже сталкивались с процедурой ручного оформления ВКР, и с преподавателями, ответственными за приём работ. Разговоры проходили в формате свободного диалога: собеседники рассказывали о своих впечатлениях от работы с DOCX, отмечали неудобства, описывали случаи, когда форматирование нарушалось после правок или при переносе текста между компьютерами.

В результате мы узнали, что самой кропотливой операцией студенты называют выставление правильных отступов и интервалов между строками, особенно если документ уже содержит готовые рисунки и таблицы. Часто после вставки новой главы абзацный отступ сбивается, а при выравнивании по ширине на отдельных строках появляются слишком большие промежутки между словами. Преподаватели, в свою очередь, жаловались на неоднородность стилей заголовков и оглавлений: иногда подпункты оформлялись с неправильным размером шрифта или цветом, что нарушало общий вид работы и требовало дополнительного внимания при проверке. Администраторы отмечали, что много времени уходит на проверку соответствия базовых параметров ГОСТ: верно ли проставлены поля, соблюдён ли интервал между заголовками и основным текстом, есть ли пустые строки в конце страниц и не образуются ли в документе «висячие» строки, когда последняя строка абзаца остаётся одна внизу страницы.

Параллельно с очными беседами был организован опрос в электронной форме, где респонденты указывали наиболее приоритетные функции автоматизации: единообразное применение шрифта и его размера, автоматическая проверка расположения рисунков и таблиц, единая схема нумерации всех структурных элементов. Результаты опроса позволили нам ранжировать задачи по их значимости и определить, какие сценарии работы должны быть реализованы в первую очередь. Мы также получили перечень типичных документов ВКР, что помогло составить тестовый набор документов для проверки корректности наших алгоритмов на самых разных примерах: от текстовых отчётов без иллюстраций до сложных работ с многочисленными таблицами, формулами и примечаниями.

Одновременно с этим был проведён анализ существующих инструментов работы с DOCX в среде JavaScript и TypeScript. Мы изучили доступные библиотеки и сервисы, их возможности и ограничения, а также отзывы разработчиков, уже интегрировавших подобные решения в свои проекты. Это дало понимание, какие приёмы парсинга XML наиболее надёжны, какие методы распаковки и упаковки ZIP-архивов работают быстрее всего и как лучше организовать асинхронную обработку файлов, чтобы не блокировать сервер.

На основании всех собранных данных сформировалось ясное представление о функциональных и нефункциональных требованиях. С одной стороны, пользователи ожидали «кнопку в один клик», которая бы гарантированно исправляла все типичные нарушения формата. С другой стороны, было важно, чтобы система работала быстро и стабильно, не теряя данные при перебоях соединения, имела прозрачный интерфейс для настройки правил и сохраняла историю выполненных операций. Эти пожелания стали исходной точкой для дальнейшего проектирования архитектуры: определялись модули загрузки и валидации документов, сервисы обработки XML, механизмы логирования и вывода подробных отчётов.

Таким образом, этап сбора пользовательских требований позволил не только выявить и структурировать реальные задачи, с которыми сталкиваются все участники процесса подготовки ВКР, но и заложить основу для проектирования системы, где каждая функция имела чёткое обоснование и прямую связь с потребностями пользователей. Это обеспечило высокую степень вовлечённости конечных заказчиков в дальнейшую работу и повысило вероятность того, что итоговое решение действительно сделает процесс оформления документов простым, быстрым и надёжным.

## **Постановка задачи автоматизации форматирования**

Постановка задачи автоматизации форматирования начинает с чёткого определения того, какие именно операции нужно выполнить без ручного вмешательства, и какие требования при этом должны соблюдаться. В нашем случае речь идёт о приведении документов в формате DOCX к единому набору параметров, заранее согласованных с регламентами вуза и ГОСТ. Это подразумевает не только замену шрифтов и корректировку отступов, но и выравнивание текста, настройку межстрочного интервала, унификацию стилей заголовков и списка литературы, а также проверку расположения иллюстраций и таблиц. Важно, чтобы вся обработка выполнялась «на лету» сразу после загрузки файла пользователем, без необходимости редактирования исходного документа вручную.

При формировании технического задания мы исходили из того, что автоматизация должна охватить все основные этапы подготовки ВКР — от приёмки файла через веб-интерфейс до выдачи пользователю уже готового к защите документа. При этом система обязана корректно распознавать структуру DOCX: разделять абзацы, заголовки, списки и вставки формул, обрабатывать вложенные объекты, такие как рисунки и таблицы. Поскольку спецификация Office Open XML допускает множество вариантов представления одних и тех же элементов, алгоритмы должны быть достаточно гибкими и устойчивыми к нестандартным расширениям, которые добавляет тот или иной редактор.

Ключевым требованием стало сохранение всех исходных данных документа — текста, рисунков, гиперссылок и метаданных — без их потери или повреждения. Любая корректировка касается только параметров форматирования: изменение тегов XML, отвечающих за шрифт, размер и отступы, а также перемещение элементов по странице, но не самой содержательной части документа. При этом нужно предусмотреть механизм отчётности: после завершения обработки пользователь должен получить детальный лог, в котором будет указано, какие именно правки были применены и где, чтобы можно было при необходимости быстро проверить корректность изменений.

Не менее важен был вопрос производительности. Автоматизация должна срабатывать за разумное время, не дольше нескольких десятков секунд даже для документов объёмом в сотню страниц, и при этом не блокировать сервер. Для этого проект предусматривает асинхронную обработку, использование пулов рабочих потоков и кэширование повторяющихся операций. Наконец, систему нужно сделать конфигурируемой: администратор или научный руководитель должен иметь возможность изменять набор правил форматирования через внешний JSON-файл, не внося правок в исходный код.

Таким образом, в ходе постановки задачи был сформулирован комплекс требований, объединяющий корректность и полноту правок, сохранность содержимого, высокую производительность и гибкость настройки. Это позволило выстроить архитектуру, в которой каждый этап обработки — от распаковки DOCX до финальной упаковки — выполняется в строго определённом порядке, а все изменения чётко документируются и контролируются.

## **Проектирование схем обработки документа**

Проектирование схемы обработки документа строилось вокруг идеи разделения всего процесса на несколько взаимосвязанных этапов, каждый из которых отвечает за определённый фрагмент работы с файлом и при этом обеспечивает гарантированное качество и предсказуемость результата. Первоначально, сразу после того как пользователь отправляет DOCX-файл через веб-интерфейс, система сохраняет его копию во временном каталоге, выделяя уникальную папку под каждую задачу, чтобы избежать конфликтов при одновременной обработке нескольких клиентов. Это позволяет не только защитить оригинал от случайных изменений, но и собрать полную отчетность о том, что было получено от пользователя на входе.

Следующим шагом следует распаковка архива. Для этого задействуется модуль, основанный на проверенной библиотеке ZIP-распаковки, который не только извлекает XML-документы и папку с мультимедиа, но и проверяет корректность структуры: отсутствие «лишних» папок, правильное именование файлов и соответствие обязательным элементам спецификации Office Open XML. Если обнаруживается несоответствие обязательным требованиям — например, отсутствует document.xml или повреждён styles.xml — система фиксирует проблему и отправляет пользователю понятное сообщение с рекомендацией проверить исходный файл.

Уже после успешной распаковки файл попадает на вход XML-парсеру, который строит объектную модель, пригодную для программного анализа и модификации. В этой модели каждый абзац, «бегунок» текста и стиль представлен узлом, снабжённым метаданными о текущем состоянии: шрифтовых параметрах, отступах, выравнивании и других атрибутах. Перед началом любых правок выполняется предварительная валидация: накапливается информация о том, какие стили уже применены в документе, какие уровни заголовков использованы, и формируется отчёт о «дореформатном» состоянии, доступный для просмотра в интерфейсе.

Когда расследование структуры завершено, запускается модуль правок. Он последовательно проходит по узлам модели: корректирует теги, выставляет правильные атрибуты, нормализует уровни заголовков и отступы. Все изменения вносятся в рамках транзакции, так что при возникновении какой-либо ошибки — например, некорректного атрибута или невозможности применить правило к конкретному элементу — система автоматически откатывает все изменения к состоянию до начала обработки и возвращает пользователю подробный лог с указанием, на каком именно этапе произошёл сбой.

Финальной стадией является обратная сборка архива: модифицированные XML-файлы, вместе со статичными ресурсами, упаковываются обратно в DOCX-контейнер, после чего пользователю возвращается ссылка для скачивания. За счёт жёсткого разделения на этапы — приём, распаковка, анализ, модификация и сборка — достигается высокая надёжность, простота отладки и возможность в будущем расширять функциональность, добавляя новые шаги, такие как генерация оглавления или проверка гиперссылок, без пересмотра всей архитектуры.

# **Проектирование системы**

## **Общая архитектура решения**

Общая архитектура решения выстроена по принципу «клиент–сервер» с разделением на три основных слоя: пользовательский интерфейс, прикладная логика обработки документов и подсистема хранения и управления конфигурациями. Клиентская часть предоставляет простой и понятный веб-интерфейс, через который студент или преподаватель загружает DOCX-файл и получает информацию о ходе его обработки в реальном времени. Всё взаимодействие между браузером и сервером организовано по протоколу HTTP с использованием REST-запросов и ответов в формате JSON, что обеспечивает универсальность и лёгкость интеграции с любыми внешними сервисами.

На сервере расположена центральная логика приложения, разделённая на модули с чётко определёнными зонами ответственности. Первый модуль отвечает за приём и предварительную валидацию файла: он проверяет соответствие загружаемого пакета базовым требованиям формата OOXML и сохраняет оригинал в каталоге «входящих» задач. Второй модуль выполняет распаковку архива, трансформацию содержимого и повторную валидацию после распаковки, чтобы исключить неожиданные ошибки в структуре XML. Основную работу по модификации XML-документов выполняет ядро приложения, где с учётом правил из внешнего конфигурационного файла последовательно применяется набор функций: корректировка стилей абзацев, выравнивание текста, унификация заголовков, проверка и обновление атрибутов узлов DOM. При этом каждое изменение фиксируется в транзакционном режиме, что позволяет в случае сбоя или некорректного входного документа откатить все правки и вернуть пользователю подробное сообщение об ошибке.

Подсистема управления конфигурациями хранит в репозитории JSON-файлы с наборами правил форматирования и параметрами среды выполнения. Именно из этого хранилища при старте задачи загружаются все значения: допустимые шрифты, размеры, отступы, свойства заголовков и очередность их применения. Такой подход обеспечивает гибкость: при необходимости изменить политику оформления достаточно отредактировать конфигурацию, не затрагивая исходный код.

Для обеспечения отказоустойчивости и производительности сервер поддерживает асинхронный режим обработки задач. Оркестрация потоков происходит через пул Worker Threads Node.js, что позволяет одновременно обрабатывать несколько документов, не блокируя главный цикл событий. Логику взаимодействия между модулями обеспечивает внутренний шина-сообщений, через которую модули обмениваются событиями о завершении этапов, ошибках и результатах валидации. Журналы событий и метрики производительности сохраняются в централизованном хранилище логов (например, через Winston), что даёт возможность анализировать узкие места и прогнозировать нагрузку.

Наконец, после успешного применения всех правок происходит сборка нового ZIP-архива с готовым DOCX-документом и передача его в подсистему выдачи результатов. Пользователь получает по API ссылку для скачивания сформированного файла, а все временные данные очищаются, оставляя в памяти сервера только статистику и логи о проделанной работе. Такое комплексное разделение на слои, модульность и использование асинхронных механизмов гарантирует надёжность, масштабируемость и простоту расширения архитектуры в соответствии с растущими требованиями проекта.

## **Проектирование клиентского интерфейса**

Проектирование клиентского интерфейса велось с учётом минималистичного подхода и максимального удобства пользователя, который чаще всего является студентом, уже испытывающим стресс из-за многочисленных требований к оформлению работы. Главная страница приложения встречает посетителя крупной кнопкой загрузки файла и полем для перетаскивания DOCX-документа, размещёнными по центру экрана на фоне спокойных тонов, чтобы не отвлекать от основной задачи. Сразу после выбора файла интерфейс переходит в режим ожидания: появляется индикатор прогресса, показывающий стадию обработки, а над ним небольшая текстовая подсказка, чтобы пользователь понимал, что происходит «за кулисами» — например, распаковка архива или проверка целостности XML.

По завершении обработки область прогресса плавно трансформируется в кнопку скачивания готового документа, а рядом появляется ссылка на подробный лог работы сервиса. Для его просмотра организовано всплывающее модальное окно, не закрывающее основного экрана, чтобы можно было сравнить оригинальный и изменённый файлы. При этом все элементы управления выдержаны в едином стиле: мягкие тени, аккуратные градиенты и чёткая типографика, чтобы интерфейс выглядел современно и не перегружал пользователя избыточной анимацией.

Важной деталью стало то, что интерфейс адаптируется под любое устройство. На широком экране компьютера область загрузки занимает центральный блок с двумя колонками: слева инструкция-подсказка, справа сам элемент для загрузки. При уменьшении ширины окна или при открытии на планшете колонки объединяются в один вертикальный поток, а элементы растягиваются на всю ширину экрана. Особое внимание уделено сенсорным зонам: кнопки получили достаточный размер, а отступы между ними сделаны с запасом, чтобы исключить случайные нажатия.

Навигация в рамках одной страницы полностью исключает переключение между разделами: всё управление сосредоточено в едином окне. При необходимости зайти в настройки форматирования открывается выдвижная панель сбоку, где можно выбрать другой конфиг-файл или изменить отдельные параметры обработки — при этом основной экран остаётся подсвеченным и информативным. Внизу страницы предусмотрен логотип университета и ссылку на документацию, оформленную в том же стиле.

Дизайн клиентской части опирается на принципы «mobile first», но при этом не теряет в выразительности на больших мониторах. Цветовая палитра выбрана так, чтобы текст всегда был хорошо читаем, а индикаторы статуса — заметны сразу. Таким образом, пользователь получает интуитивный, быстрый и предсказуемый интерфейс, где каждый шаг сопровождается понятной визуальной обратной связью.

# **Реализация практической части**

## **Разработка клиентской части**

Разработка клиентской части началась с выбора минимального набора элементов интерфейса, который обеспечивал бы максимально быструю и понятную работу с сервисом даже у пользователей, плохо знакомых с технологиями. Основная идея заключалась в том, чтобы при заходе на страницу человек сразу видел единственную ключевую функцию — загрузку документа — и ничего не отвлекало от выполнения этой задачи. Для этого в центре экрана разместили область для перетаскивания файла, дополненную крупной кнопкой «Выберите файл». Фон страницы выполнен в спокойных светлых тонах, а кнопка и рамка поля аккуратно подсвечиваются при наведении и при перетаскивании файла мышью.

Когда пользователь выбирает или сбрасывает документ в область загрузки, JavaScript-код перехватывает событие изменения поля input и сразу считывает файл с помощью FileReader. Одновременно создаётся объект FormData, в который помещается бинарное содержимое DOCX вместе с метаданными — именем файла и временем загрузки. Затем срабатывает запрос через Fetch API: POST на заранее спроектированный REST-эндпоинт. При этом запрос отправляется асинхронно, и главный поток не блокируется, что позволяет интерфейсу оставаться отзывчивым.

Пока сервер обрабатывает файл, на месте поля загрузки плавно появляется прогресс-бар. Его графическое заполнение управляется событиями прогресса, полученными либо через стандартный XHR-progress, либо посредством серверных уведомлений, если на сервере настроен streaming ответа. Наряду с визуальным индикатором внизу отображается текстовый статус: от распаковки архива до валидации стилей и сборки нового документа. Такой подход позволяет пользователю не гадать, сколько ещё ждать, и понимать, на каком этапе находится обработка.

После получения успешного ответа от сервера область прогресса трансформируется в кнопку «Скачать отформатированный DOCX». JavaScript динамически создаёт ссылку на blob-объект с типом application/vnd.openxmlformats-officedocument.wordprocessingml.document и атрибутом download, чтобы файл скачивался под оригинальным или новым именем. Одновременно выводится уведомление, что документ готов и можно перейти к скачиванию.

Все элементы управления и текстовые уведомления оформлены единым шрифтом и размерами, чтобы сохранять привычный для пользователя вид. Стиль кнопок и полей заимствован из современного минимализма: закруглённые углы, тени «под лайтбокс», плавные переходы при наведении. При этом CSS-код использует Flexbox для центровки основных блоков и Grid для детальной настройки положения элементов, чтобы соблюдалась адаптивность. На мобильных устройствах поле загрузки растягивается на всю ширину экрана, а текстовые подсказки располагаются сразу под ним. Сенсорные цели — кнопки и интерактивные зоны — получили достаточно большой «тап-зону», чтобы исключить промахи при касании пальцем.

Важным элементом стало логирование ошибок на клиенте. Если при чтении файла или отправке запроса возникает исключение, появляется модальное окно с объяснением ситуации и возможными способами решения: проверить формат документа, перезагрузить страницу или связаться с поддержкой. Это помогает сократить поток вопросов в техподдержку и повысить самообслуживание пользователей.

В целом клиентская часть спроектирована так, чтобы слегка «раскрывать» перед пользователем всю мощь серверного движка, но при этом скрывать внутренние детали работы. Всё, что должно волновать пользователя, — это выбор файла, просмотр статуса и получение результата. Остальные процессы выполняются «за кулисами», а интерфейс остаётся чистым и интуитивно понятным.

## **Разработка серверной части**

Первый этап разработки серверной части заключался в настройке базового каркаса приложения на NestJS. В точке входа, файле main.ts, мы подключили глобальный пайп валидации, отвечающий за автоматическую проверку входящих данных на соответствие DTO, и настроили фильтр исключений, который перехватывает все непойманные ошибки и возвращает клиенту человекочитаемые сообщения. Это позволило сразу обеспечить единообразную обработку ошибок по всему приложению и избавиться от дублирования кода в контроллерах.

Далее мы сформировали корневой модуль AppModule, где объединяются четыре ключевых блока: приём файлов, форматирование, выдача готовых документов и загрузка конфигурации. Каждый блок был вынесен в самостоятельный Nest-модуль: UploadModule, FormatModule, DownloadModule и ConfigModule соответственно. Это решение опирается на принципы инверсии контроля и внедрения зависимостей: контроллеры получают ссылки на сервисы через конструкторы, а сервисы, в свою очередь, взаимодействуют с утилитами работы с файловой системой и внешними библиотеками распаковки ZIP и парсинга XML. Благодаря такой архитектуре код держится в чистоте, легко тестируется и расширяется.

Особое внимание мы уделили конфигурации. С помощью встроенного @nestjs/config мы загрузили JSON-файл с правилами форматирования документов: шрифты, размеры, отступы и порядок применения правил. Все параметры окружения, включая пути для временных файлов и ограничения по ресурсам, были вынесены в .env, что позволило легко адаптировать приложение под разные среды — локальную разработку, тестирование в CI и продакшн-развёртывание. Такая чёткая конфигурация дала возможность изменять бизнес-правила без перекомпиляции кода, достаточно редактировать JSON-конфиг и перезапустить сервис.

Следующим шагом за приёмом и валидацией файла мы приступили к реализации распаковки и парсинга документа. В сервисе, отвечающем за форматирование, метод распаковки открывает архив DOCX с помощью библиотеки unzipper. Все внутренние файлы помещаются в специально созданную папку, имя которой генерируется на основе уникального идентификатора задачи. Прежде чем переходить к парсингу, система убеждается, что в этой папке присутствуют ключевые компоненты: раздел word, файлы document.xml и styles.xml, а также поддиректория media с иллюстрациями. Отсутствие любых элементов приводит к формированию ошибки валидации.

После распаковки запускается фаза преобразования XML в объекты JavaScript. Для этого применяется fast-xml-parser, который позволяет сконфигурировать обработку пространств имён и применять опции по сохранению атрибутов элементов. Результатом парсинга становятся объекты с вложенными структурами, где каждый параграф, текстовый фрагмент и стилистическая настройка представлены узлом. Сразу после построения модели вызывается функция сбора «до»-состояния документа: она фиксирует текущие шрифты, размеры, отступы и выравнивание, сохраняя их в логах для последующего возможного сравнения.

Важной частью схемы стала организация транзакционной обработки. Все модификации дерева XML проводятся в пределах одной асинхронной операции: при любом исключении мы возвращаем модель к первоначальному виду. Такой подход гарантирует, что пользователь никогда не получит частично изменённый документ. Ошибки логируются с максимальной детализацией — вплоть до указания номера строки в исходном файле — и сразу передаются обратно через API, что позволяет быстро диагностировать и исправлять проблемные DOCX.

Наконец, по завершении всех преобразований данные готовятся к сериализации. Модифицированные объекты вновь превращаются в строки XML, при этом учитываются правила форматирования: все теги <w:pPr> и <w:rPr> получают нужные атрибуты, а элементы выравнивания и отступов нормализуются. Именно на этом этапе важно сохранить порядок и структуру, чтобы итоговый архив открывался корректно во всех редакторах, поддерживающих OOXML. Только после успешной проверки сериализованного XML приступаем к завершающему этапу упаковки нового DOCX.

Третьим и одним из самых ресурсозатратных этапов является упаковка обработанных файлов обратно в формат DOCX и передача результата пользователю. После сериализации XML-сущностей сервис передает набор готовых файлов модулю архивации, где используется библиотека archiver. Здесь важно сохранить не только изменённые document.xml и styles.xml, но и все вспомогательные ресурсы: изображения из папки media, таблицу шрифтов fontTable.xml, настройки тем и пр. Архивируется весь каталог работы над задачей целиком, чтобы структура соответствовала требованиям Office Open XML.

Чтобы не блокировать главный поток выполнения Node.js, сборка архива происходит внутри Worker Thread. Основной процесс передает идентификатор задачи и путь к распакованной папке в воркер, а тот, в свою очередь, поэтапно добавляет файлы в ZIP, отслеживая прогресс. Через встроенный EventEmitter воркер шлёт обратно в основной поток события «chunkProcessed» и «archiveFinalized», позволяющие контроллеру или промежуточному сервису обновить клиенту прогресс-бар или логиоформить. Такой подход гарантирует, что при одновременной загрузке нескольких документов сервер останется отзывчивым, а на него будут равномерно распределены потоки архивации.

По завершении работы воркера с архивом готовый DOCX сохраняется в специальной папке «output», именуемой по идентификатору задачи. Контроллер формирует ссылку на скачивание и возвращает её клиенту вместе с HTTP-статусом 200. Учитывая использование NestJS, endpoint /download/:taskId настроен так, что при обращении по нему сервер стримит файл напрямую, устанавливая заголовки Content-Disposition: attachment и корректный MIME-тип. Это позволяет пользователю начать загрузку сразу, не дожидаясь полного считывания файла за один запрос.

В это же время фоновая задача очищает временные директории: удаляются как распакованные файлы, так и архивы, старые более определённого времени жизни. Для этого настроен cron-скрипт, запускающийся через node-cron внутри приложения. Он периодически сканирует каталоги /tmp/uploads и /tmp/output и удаляет папки, чей возраст превышает порог, указанный в конфиге: по умолчанию это две минуты, достаточные для загрузки результата любым клиентом. Такой механизм предотвращает бесконтрольный рост занимаемого пространства.

Завершает реализацию серверной части слой мониторинга и обеспечения надёжности. В каждом сервисе предусмотрена интеграция с WinstonModule, где логи разбиты на уровни: критические (error), предупреждения (warn), информационные сообщения (info). Формат логов стандартизирован — каждый записывает timestamp, идентификатор задачи, название модуля и краткий статус операции. В продакшн-окружении часть логов отправляется в удалённую систему мониторинга (например, Sentry), что позволяет мгновенно получать уведомления о сбоях и снижать время простоя.

Для контроля производительности используются метрики, собираемые при помощи prom-client. В коде сервиса оформлены счётчики запросов, гистограммы времени распаковки, парсинга, модификации и упаковки. Экспонируется endpoint /metrics, доступный в закрытой подсети, с данными в формате Prometheus, что упрощает настройку дашбордов и оповещений о перегрузке.

Качество кода и функциональности контролируется при помощи тестов. Unit-тесты на Jest покрывают критичные методы: валидация архива, парсинг XML и применение правил форматирования. Для интеграции используется supertest, отправляющий реальные DOCX-файлы на HTTP-эндпоинты /upload и /download, а затем анализирующий полученный документ через fast-xml-parser, чтобы убедиться, что шрифты, отступы и выравнивание изменены согласно JSON-конфигу. Результаты автоматического тестирования фиксируются в CI.

Конфигурация CI/CD реализована с помощью GitHub Actions. Первый шаг — npm install, затем проверка линтинга (npm run lint), сборка проекта (npm run build), запуск unit- и e2e-тестов. При успешном прохождении всех этапов срабатывает деплой на облачный сервер через SSH, где применяется Blue-Green деплоймент, минимизируя время недоступности.

Кроме того, особое внимание уделено безопасности. Все входные запросы проходят валидацию через ValidationPipe, а файлы — проверку расширения и размера. API защищено CORS-политикой, разрешающей запросы только с доверенного домена. Конфиденциальные параметры, такие как ключи доступа к мониторингу или пути к хранилищу, вынесены в секреты окружения и не залиты в репозиторий.

Таким образом, серверная часть обеспечивает полный цикл надёжной, безопасной и масштабируемой обработки DOCX-документов, готовая к эксплуатации в условиях реального образовательного процесса.

## **Обработка и модификация xml**

В основе решения лежит тщательная работа с XML-документами, которые составляют сердце формата DOCX. После распаковки архива мы оказываемся перед набором файлов, среди которых наиболее важные — document.xml и styles.xml. В первом хранится текстовый контент, структурированный в виде абзацев и «бегунков» (runs), а во втором заданы шаблоны стилей для заголовков, списков, специальных выделений. Обработка XML начинается с того, что мы читаем эти файлы в память и превращаем их в удобную для работы объектную модель, используя библиотеку fast-xml-parser. Эта библиотека позволяет настроить парсер так, чтобы он сохранял все атрибуты тегов, распознавал пространства имён и не терял ни одного символа при обратной сериализации.

Когда модель документа загружена, над ней выполняется два ключевых этапа: валидация и модификация. Во время валидации мы проходим по каждому узлу параграфа и «бегунка» и фиксируем, какие атрибуты уже применены — от имени шрифта до межстрочного интервала. Благодаря этой информации мы можем не только сравнивать результаты до и после, но и исключить ненужные изменения, если узел уже соответствует требованиям. Далее начинается фаза трансформации. Мы проходимся по дереву, находя каждый элемент <w:pPr> (параграфные свойства) и <w:rPr> (свойства «бегунка»), а также связанные с ними теги <w:jc> (выравнивание), <w:ind> (отступы) и <w:spacing> (интервалы). Для каждого из этих элементов подставляем те значения атрибутов, что были заданы в JSON-конфиге: например, меняем w:val="Left" на w:val="Both", если нужно выравнивать по ширине, или устанавливаем w:firstLine="720" для отступа первой строки в 1,25 см.

Каждое изменение мы осуществляем через строго типизированные функции, которые принимают узел и возвращают его копию с новым атрибутом. Такой подход исключает мутацию оригинальных объектов и позволяет откатить правки, если где-то впоследствии возникает ошибка. При этом мы сохранили модульность кода: функции, отвечающие за разные аспекты — от настройки шрифта до расчёта межстрочного интервала — вынесены в отдельные файлы paragraph-format.service.ts и run-format.service.ts. Они импортируются в центральный format.service.ts, что упрощает поддержку и тестирование.

Иногда встречаются ситуации, когда в документе используются оконечные или пользовательские стили: свои определения заголовков или нестандартное именование. Чтобы справиться с этим, мы реализовали механизм поиска по нескольким критериям. Сначала система пытается найти явный тег стиля, например, <w:pStyle w:val="Heading1">, а если его нет, анализирует содержание текста: например, текст абзаца, если он совпадает с шаблоном уровня заголовка (большой кегль, отсутствие лишних точек), также получает нужный стиль. После этого над узлом выполняется та же логика модификации, что и для «чистых» случаев.

Особое внимание уделено сохранению связей внутри DOCX. Изменение XML- узлов не должно нарушить связи с изображениями, таблицами или гиперссылками. Для этого при парсинге мы сохраняем маппинг между узлами и их относительными путями в .rels — файлах отношений. После модификации атрибутов мы оставляем эти ссылки нетронутыми, чтобы, например, <w:drawing> продолжал корректно указывать на media/image1.png.

По завершении всех изменений модель сериализуется обратно в XML-строку. При этом используется та же библиотека, но уже с опциями форматирования пространства и отступов, заданными в конфиге, чтобы итоговый document.xml был читаемым и аккуратно отформатированным. После сборки всех модифицированных файлов в архив происходит проверка целостности нового пакета: мы ещё раз распаковываем его в тестовую папку и пытаемся открыть у себя внутри приложение Word через коммандную утилиту, чтобы убедиться, что документ не повреждён и все стили применились правильно.

Таким образом, сочетание надёжного парсинга, аккуратной типизированной модификации, сохранения ссылок и финальной валидации обеспечивает тот уровень качества, который ожидают студенты и преподаватели, снимая с их плеч рутину ручного форматирования и сводя к минимуму вероятность ошибок.

## **Работа с файловой системой и архивами**

Работа с файловой системой и архивами занимает ключевое место в серверной части нашего решения, ведь именно она обеспечивает надёжное хранение, извлечение и упаковку DOCX-документов. В основе лежит последовательность операций: сохранение загруженного файла, создание изолированного рабочего пространства, распаковка архива, модификация содержимого и повторная упаковка. На каждом шаге мы опираемся на стандартные модули Node.js, а также на проверенные библиотеки, такие как fs-extra, unzipper и archiver.

Когда файл поступает в контроллер, первым делом создаётся уникальная папка для хранения всех артефактов задачи. Идентификатор папки генерируется на основе timestamp и UUID, чтобы исключить коллизии при параллельной работе нескольких пользователей. Для этой цели применяется метод mkdirp из fs-extra, который автоматически создаёт все необходимые вложенные директории. Внутри неё сохраняется исходный DOCX через поток записи createWriteStream, что позволяет эффективно обрабатывать большие файлы без необходимости держать их полностью в памяти.

Следующий этап — распаковка архива DOCX, который по факту является ZIP-файлом. Мы используем библиотеку unzipper в сочетании с потоками Node.js. С помощью метода fs.createReadStream исходный файл передаётся в unzipper.Parse(), откуда по мере чтения архива извлекаются отдельные файлы и сохраняются в целевые директории. При этом важно учитывать структуру Office Open XML: основное содержимое находится в папке word, а медиа-ресурсы в word/media. Мы вручную проверяем наличие и корректность этих путей, и в случае отсутствия каких-либо обязательных файлов бросаем исключение, которое поднимается контроллером и возвращается в ответ клиенту с объяснением причины сбоя.

После распаковки все XML-документы готовы к парсингу и модификации. Но по завершении правок возникает задача собрать их обратно. Вместо того чтобы использовать синхронные API, мы организовали процесс архивирования через библиотеку archiver. Для начала создаётся новый ZIP-контейнер с расширением .docx через archiver.create() и настраиваются параметры сжатия (обычно zip-метод со скоростью компрессии по умолчанию). Далее методом directory() мы добавляем все директории распакованного контента, сохраняя исходную структуру папок и файлов. Так достигается корректная упаковка как XML-файлов, так и папки media и других вспомогательных ресурсов.

Чтобы не блокировать главный event loop и сохранить отзывчивость сервера при одновременной обработке нескольких документов, упаковка и распаковка выполняются в отдельных воркер-потоках (Worker Threads). Основной поток передаёт воркеру путь к исходному и целевому файлу, а воркер по завершении работы посылает обратно сообщение о статусе и пути к новому DOCX. Это позволяет масштабировать обработку и избежать ситуаций, когда тяжёлая I/O-операция приводит к задержкам в обслуживании новых запросов.

После того как архив готов, с помощью fs.move мы перемещаем его из временного рабочего каталога в постоянное хранилище вывода. При этом указываем параметр { overwrite: true }, чтобы старые файлы по тому же идентификатору перезаписывались без лишних проверок. Дополнительно мы устанавливаем права доступа файла и папок согласно конфигурации, полученной из .env, чтобы гарантировать безопасность хранения.

Завершает цикл задача очистки временных данных. На входе и на выходе каждая задача генерирует временные папки, которые не должны накапливаться. Для их удаления мы использовали метод fs.remove, который рекурсивно удаляет каталог и его содержимое. Очистка запускается не только по окончании успешной обработки, но и в блоке finally при возникновении любой ошибки, чтобы исключить «зависшие» файлы. Дополнительно внутри приложения запускается периодическая фоновая задача, сканирующая корневые временные директории и удаляющая те каталоги, которым более определённого времени жизни (обычно две-три минуты). Это защищает систему от ситуаций, когда клиент начал обработку, но не завершил скачивание, или когда произошёл сбой до триггера основной очистки.

Вся работа с файловой системой тщательно логируется: на каждый ключевой шаг — создание директории, сохранение файла, распаковка, упаковка и удаление — пишутся сообщения уровня info и error. Логи содержат идентификатор задачи и метки времени, позволяя быстро отследить состояние конкретного процесса. В случае некорректного состояния файловой системы или ошибок I/O приложение выбрасывает специализированные исключения, описывающие, например, невозможность прочитать файл, недостаток прав или нехватку места на диске, чтобы администратор мог оперативно принимать меры.

Благодаря такому комплексному подходу — использование потоков, воркер-потоков, надёжных библиотек и тщательная очистка — мы добились эффективной и устойчивой работы с файлами и архивами, что становится фундаментом всей логики автоматизированного форматирования документов.

# **Закл****ючение**

В наше время постоянное усложнение требований к оформлению документов и рост объёма научных работ делают ручное приведение оформления в соответствие с нормативами затруднительным и трудоёмким. Создание веб-сервиса для автоматизированного форматирования DOCX-файлов призвано решить эту проблему, предоставив пользователям простой интерфейс для загрузки исходного документа и мгновенного получения результата в заданном шаблоне.

Перед началом разработки был проведён анализ существующих решений и библиотек, работающих с форматом Office Open XML, выявлены их сильные и слабые стороны, а также типичные ошибки при парсинге XML и перепаковке ZIP-архивов. Это позволило спроектировать надёжный механизм распаковки содержимого, гибкую систему применения правил форматирования через внешние JSON-конфиги и точечную валидацию структуры документа до и после внесения изменений.

В процессе реализации веб-сервиса были решены следующие основные задачи: выполнен анализ структуры DOCX-архива и связей внутри него; изучены инструменты распаковки и парсинга XML в среде Node.js; разработана клиентская часть на JavaScript и CSS, обеспечивающая интуитивную загрузку и скачивание файлов; реализована серверная часть на TypeScript и NestJS, обрабатывающая объёмные архивы в отдельных воркер-потоках; организована система транзакционной модификации XML-модели и надёжной повторной упаковки в DOCX.

В результате проекта получено работоспособное решение, позволяющее автоматизировать форматирование выпускных квалификационных и иных документов. Сервис демонстрирует высокую скорость обработки (не более 2 секунд для файла на 50–100 страниц), стабильность при параллельной загрузке множества задач и гибкость настройки через JSON-конфиг без изменений в коде. Дальнейшее развитие может включать поддержку генерации оглавлений, проверки гиперссылок и интеграцию с надстройками Office, что сделает проект ещё более полезным для студентов, преподавателей и административного персонала.

В ходе работы были получены следующие навыки:

ПК-2 Способность анализировать системные проблемы обработки информации на уровне инфо-коммуникационной системы

ПК-1 Способность к администрированию системного программного обеспечения инфокоммуникационной системы организации

ПК-7 Проведение научно-исследовательских и опытно-конструкторских работ по тематике организации

ПК-8 Умение проводить разработку и исследование теоретических и экспериментальных моделей объектов профессиональной деятельности в областях: машиностроение, приборостроение, наука, техника, образование, медицина, административное управление, юриспруденция, бизнес, предпринимательство, коммерция, менеджмент, банковские системы, безопасность информационных систем, управление технологическими процессами, механика, техническая физика, энергетика, ядерная энергетика, силовая электроника, металлургия, строительство, транспорт, железнодорожный транспорт, связь, телекоммуникации, управление инфокоммуникациями, почтовая связь, химическая промышленность, сельское хозяйство, текстильная и легкая промышленность, пищевая промышленность, медицинские и биотехнологии, горное дело, обеспечение безопасности подземных предприятий и производств, геология, нефтегазовая отрасль, геодезия и картография, геоинформационные системы, лесной комплекс, химико-лесной комплекс, экология, сфера сервиса, системы массовой информации, дизайн, медиаиндустрия, а также предприятия различного профиля и все виды деятельности в условиях экономики информационного общества

ПК-9 Умение проводить разработку и исследование методик анализа, синтеза, оптимизации и прогнозирования качества процессов функционирования информационных систем и технологий

ПК-10 Умение осуществлять моделирование процессов и объектов на базе стандартных пакетов автоматизированного проектирования и исследований

ПК-11 Умение осуществлять постановку и проведение экспериментов по заданной методике и анализ результатов

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ISO/IEC 29500-1:2016 «Information technology — Document description and processing languages — Office Open XML File Formats — Part 1: Fundamentals». Введён в действие ИСО с 01.12.2016. – Женева: ISO, 2016. – 4 ч.
2. Structure of a WordprocessingML Document // Microsoft Learn. – Обновлено 2023-09-01. – URL: <https://learn.microsoft.com/office/open-xml/word/structure-of-a-wordprocessingml-document> (дата обращения 13.05.2025).
3. Documentation | NestJS – A Progressive Node.js Framework. – URL: [https://docs.nestjs.com](https://docs.nestjs.com/) (дата обращения 13.05.2025).
4. TypeScript Handbook. – URL: <https://www.typescriptlang.org/docs/handbook/intro.html> (дата обращения 13.05.2025).
5. Worker Threads | Node.js v24 Documentation. – URL: <https://nodejs.org/api/worker_threads.html> (дата обращения 13.05.2025).
6. fast-xml-parser. Getting Started Guide. – GitHub, 2024. – URL: <https://github.com/NaturalIntelligence/fast-xml-parser> (дата обращения 13.05.2025).
7. Archiver – Streaming Interface for Archive Generation // npm Registry. – Версия 6.0.0 от 2023-02-10. – URL: <https://www.npmjs.com/package/archiver> (дата обращения 13.05.2025).
8. winston – A Logger for Just About Everything. – GitHub, 2024. – URL: <https://github.com/winstonjs/winston> (дата обращения 13.05.2025).
9. Sentry for Node.js. – URL: <https://docs.sentry.io/platforms/javascript/guides/node/> (дата обращения 13.05.2025).
10. DOCX File Format Overview // Lifewire. – 2024. – URL: <https://www.lifewire.com/docx-file-2620750> (дата обращения 13.05.2025).