Содержание

Введение

Актуальность работы

В современной системе высшего образования и научных исследований Российской Федерации проверка соответствия научных публикаций официальному перечню рецензируемых изданий Высшей аттестационной комиссии (ВАК) [12] является критически важной задачей. Эта процедура необходима как при аттестации научных кадров, так и при оценке исследовательской деятельности вузов и научных организаций. Однако текущий процесс валидации полностью основан на ручной проверке через веб-портал, где необходимо последовательно искать каждый журнал в перечне. Это создаёт существенные трудности, а именно:

- **Отсутствие API:** перечень доступен исключительно в виде PDF-документов, что исключает возможность автоматизированной интеграции;
- **Динамичность данных:** перечень обновляется каждые 1–3 месяца, и необходимо отслеживать исторические версии для проверки старых публикаций;

Нерешенные проблемы

Анализ предметной области выявил следующие критические пробелы:

- Отсутствие стандартизированной структуры данных для хранения информации о перечнях на конкретные даты;
- Невозможность надёжного автоматического парсинга PDF из-за вариативности структуры и форматирования документов;
- Отсутствие механизма версионирования с отслеживанием изменений в составе перечня;
- Недостаток информации о точных датах включения и исключения журналов;
- Наличие аномалий в данных, затрудняющих корректную валидацию;
- Необходимость интеграции с внешними сервисами (такими как CrossRef API или Arxiv API) для получения метаданных публикаций.

Научная новизна работы

Научная новизна данного исследования заключается в преодолении фундаментальных проблем неструктурированных, динамически изменяющихся и зашумленных данных в кон-

тексте научной метрологии и информационного обеспечения науки. В отличие от существующих решений, предлагаемая работа базируется на следующих научных положениях и подходах:

1. **Разработка формальной онтологии и концептуальной модели для представления динамических нормативных перечней.** Впервые предлагается многоаспектная модель данных, которая не просто фиксирует статичный список объектов, а описывает жизненный цикл научного журнала в контексте перечня ВАК. Модель учитывает временную размерность (включение, исключение), статусную атрибутику и связи между различными версиями перечня, что позволяет однозначно определять статус журнала на любую историческую дату и решает проблему «версионного хаоса».
2. **Разработка алгоритмов временного логического вывода для валидации научных публикаций.** В работе предлагаются формальные алгоритмы, которые на основе построенной онтологии и временной метки публикации выполняют проверку её соответствия не просто актуальному списку, а той версии перечня, которая действовала на момент публикации. Этот подход решает ключевую научно-практическую проблему ретроспективной валидации, которая ранее не могла быть решена в автоматизированном режиме из-за отсутствия связанных исторических данных

Содержание по главам

Работа состоит из следующих ключевых разделов:

1. **Глава 1:** анализ существующих методов парсинга, сравнение подходов к концептуализации предметной области, выбор концептуальной модели, технологического стека и их обоснование, постановка задач для реализации.

Практическая значимость работы

Разработанная система может быть использована вузами, научными учреждениями, издательствами и организациями РАН для автоматизации и оптимизации процесса проверки соответствия публикаций перечню ВАК, что существенно снижает временные затраты и риск ошибок.

1. Анализ существующих методов парсинга данных в области документооборота

В настоящей главе представлены результаты изучения существующих методов парсинга данных в области документооборота: парсинг PDF-документов, открытых и закрытых API, способов организации семантического описания предметной области, технологий разработки веб-приложений и их сравнительный анализ применительно к задаче построения системы валидации научных публикаций по официальному перечню рецензируемых научных изданий Российской Федерации.

1.1 Изучение и сравнительный анализ методов концептуализации предметной области

Аннотация. *Раздел посвящён исследованию и сравнительному анализу методов концептуализации предметной области, применимых при проектировании системы валидации научных публикаций. Рассмотрены традиционные и современные подходы к моделированию семантических связей между сущностями (ER-диаграммы, UML, OWL-онтологии, плоские таблицы). Проведено их сравнение по критериям применимости, гибкости, сложности реализации и способности отражать временные аспекты данных. Выделены преимущества комбинированного подхода, основанного на использовании ER-модели для концептуального описания структуры базы данных и UML-диаграмм для отображения семантических и поведенческих связей. Сформулирована трёхуровневая концептуальная модель данных, обеспечивающая поддержку версионности перечней и семантических связей между сущностями «Журнал», «Специальность» и «Версия перечня».*

1.1.1 Основные подходы к выбору методологии построения концептуальных моделей

Выбор адекватной модели концептуализации предметной области является критическиважным этапом проектирования информационной системы. Для задачи валидации научныхпубликаций необходимо выбрать подход, который:

- обеспечивает эффективное хранение исторических версий перечня ВАК;
- поддерживает семантические связи между журналами, специальностями и датами включения/исключения;

– позволяет отвечать на запросы типа «Был ли журнал X в перечне на дату Y?»;
– обладает достаточной гибкостью для обработки аномалий в данных;
– интегрируется с современными системами управления базами данных;
– поддается расширению при изменении требований.

1.1.2 Методы описания концептуальной модели

Для организации извлечённых данных необходимо выбрать модель, которая позволит эффективно хранить исторические версии перечня и поддерживать семантические связи между журналами, специальностями и датами (см. табл. 1.1).

Таблица 1.1 – Сравнение методов концептуализации предметной области

Метод	Описание	Преимущества	Недостатки
ER-диаграмма	Сущности и связи(классический подход)	Простая реализация, хорошо поддерживается SQL БД, хорошо поддерживается ORM	Не отражает временные аспекты
UML-диаграмма	Классы, ассоциации, композиции, агрегация	Гибкость, поддержка наследования	Избыточна для простых случаев
OWL-онтология	Граф знаний, семантика	Расширяемость, логический вывод	Сложность реализации
Плоская таблица	Денормализованная структура	Простота, быстрый поиск	Избыточность данных, аномалии

1.1.3 Выбор метода концептуализации

На основе проведенного анализа для системы валидации выбран **комбинированный подход**:

1. **ER-диаграммы** используются для построения концептуальной модели базы данных с расширением для отслеживания версий перечня и временных интервалов включения журналов;
2. **UML-диаграммы** применяются для построения семантических связей между моделями данных, включая диаграммы классов, сценариев и деятельности, описывающими бизнес-процессы и взаимодействие пользователя с системой.

Трехуровневая концептуальная модель

Трехуровневая концептуальная модель данных обеспечивает поддержку версионности перечней и семантических связей между сущностями «Журнал», «Специальность» и «Версия перечня»:

1. **Слой версий перечня:** Сущность `JournalListVersion` хранит информацию о каждой загруженной версии перечня с указанием даты «по состоянию на» и временной метки загрузки;
2. **Слой журналов и специальностей:** Сущность `Journal` и `ScientificSpecialty` содержат базовые данные о журналах (ISSN, название, издатель, страна) и специальностях (код, категория, наименование);
3. **Слой связей:** Сущность `JournalListing` связывает версию перечня, журнал и диапазон дат включения/исключения, позволяя отслеживать статус журнала на конкретную дату;

Такая архитектура позволяет ответить на ключевой вопрос: «Был ли журнал `Journal` с ISSN `Journal.issn` = «X» включён в перечень ВАК на дату «Y»?» путём запроса к таблице `JournalListing` с условием:

```
/* Абстрактный запрос на получение
статуса журнала на конкретную дату Y */
SELECT * FROM journal_listings
WHERE version_id IN (
    SELECT id FROM journal_list_versions
    WHERE valid_date <= Y
    AND journal_id = (
        SELECT id FROM journals WHERE issn = 'X'
    )
AND date_included <= Y
AND (date_excluded IS NULL OR date_excluded > Y)
)
```

Дополнительные семантические ограничения

Модель включает следующие семантические ограничения целостности:

- **Временная согласованность:** `date_included` всегда предшествует `date_excluded`;
- **Уникальность версий:** на каждую дату может быть загружена только одна версия перечня;
- **Невозможность перекрытия:** для одного журнала в одной версии не допускается наличие пересекающихся временных интервалов включения или исключения;
- **Ссылочная целостность:** все ссылки на журналы и специальности должны указывать на существующие записи в соответствующих таблицах;

1.1.4 Выводы по разделу

Проведённый анализ методов концептуализации показал, что ER-модель в сочетании с UML-диаграммами представляет оптимальное решение для задачи проектирования системы валидации. Такой комбинированный подход обеспечивает:

- **Простоту реализации** благодаря прямому соответствию ER-модели с реляционной схемой БД;
- **Гибкость в представлении** сложных семантических отношений через UML-диаграммы;
- **Поддержку версионности** через трёхуровневую архитектуру с явным отслеживанием временных интервалов;
- **Масштабируемость** за счёт применения нормализации и минимизации избыточности.

Разработанная концептуальная модель служит основой для детального проектирования архитектуры базы данных и определяет интеграционные точки между различными компонентами системы.

1.2 Изучение и сравнительный анализ возможностей семантического описания процессов в предметной области

Аннотация. *Раздел посвящён исследованию методологических подходов к семантическому описанию бизнес-процессов валидации научных публикаций. Рассмотрены традиционные и современные методы формализации процессов: IDEF0, UML Activity Diagrams, BPMN, а также текстовое описание. Проведено их сравнение по критериям наглядности, формализма, применимости и практической ценности для данной предметной области. Выделены преимущества комбинированного подхода, основанного на использовании UML Activity Diagrams для визуальной коммуникации и BPMN для формального описания процессов.*

1.2.1 Актуальность семантического описания процессов

В контексте разработки информационной системы валидации научных публикаций критически важной является четкая формализация и документирование бизнес-процессов. Семантическое описание процессов решает следующие задачи

- **Требования к функциональности:** служит основой для формулирования и верификации функциональных требований к системе;
- **Архитектурное проектирование:** определяет структуру модулей, интеграционные точки и потоки данных;
- **Тестирование и валидация:** предоставляет сценарии для разработки тестовых случаев и проверки корректности реализации;
- **Документирование:** создает базис для подготовки пользовательской документации и руководств.

1.2.2 Сравнительный анализ методов формализации процессов

Для описания процесса валидации научной статьи рассмотрены следующие подходы (см. табл. 1.2):

Таблица 1.2 – Сравнение методов формализации процессов

Метод	Применение	Наглядность	Формализм
IDEF0	Функциональное моделирование	Средняя	Высокий
UML Activity Diagrams	Визуальное моделирование процессов	Высокая	Средний
BPMN (Business Process Model and Notation)	Бизнес-процессы, управление потоками	Очень высокая	Высокий
Текстовое описание	Неформальное описание процессов	Низкая	Низкий

Для системы валидации научных публикаций оптимальным является комбинированный подход:

1. **UML Activity Diagrams** используются для визуализации основных процессов и сценариев использования, обеспечивая ясное понимание потоков данных и взаимодействия компонентов системы [15, 6, 11];
2. **BPMN** применяется для формального описания критических и сложных процессов [6, 11, 14], особенно при необходимости интеграции с внешними системами (CrossRef API, системы вузов);

3. **Текстовое описание** используется для дополнительного описания процессов и сценариев использования, обеспечивая гибкость и универсальность при отсутствии необходимости визуального представления.

1.2.3 Выводы по разделу

Проведенный анализ показал, что для эффективного описания процессов в системе валидации необходимо применять комбинированный подход, сочетающий достоинства различных методологий. UML Activity Diagrams обеспечивают необходимую наглядность для коммуникации [15], BPMN предоставляет формальность для реализации [6, 14], а детальные сценарии использования служат мостом между концептуальным описанием и практической реализацией. Такой комплексный подход гарантирует, что все заинтересованные стороны имеют единое понимание функциональности и требований системы, что существенно снижает риск неправильной интерпретации требований и ошибок в реализации.

1.3 Сравнительный анализ серверных технологий для веб-приложений

Аннотация. *В разделе представлен анализ современных серверных технологий и языков программирования, применимых для разработки backend-части системы. Рассмотрены критерии выбора: универсальность языка, наличие библиотек для парсинга PDF, работа с внешними API, поддержка асинхронности и зрелость экосистемы. Проведён анализ выбора языка программирования с акцентом на задачу парсинга и активной работы с БД. На основании анализа обоснован выбор языка Python благодаря широкому набору специализированных библиотек и зрелой инфраструктуре для построения REST API. Выбран фреймворк FastAPI как оптимальный вариант для асинхронной и масштабируемой реализации сервиса,обеспечивающий автоматическую валидацию данных и высокую производительность.*

Для реализации серверной части системы валидации научных статей необходимо выбрать адекватный язык программирования и веб-фреймворк. Выбор обусловлен спецификой задачи: парсинг PDF, работа с внешними API, обработка структурированных данных, асинхронная обработка запросов.

1.3.1 Выбор языка программирования

На основе анализа предметной области и архитектурных требований сформулированы следующие критерии выбора языка:

– **Универсальность и объектно-ориентированность:** возможность разработки как консольных утилит (парсеров), так и веб-сервисов;

- **Наличие библиотек для парсинга данных:** готовые решения для работы с PDF;
- **Качественные веб-фреймворки:** наличие проверенных и хорошо документированных фреймворков;
- **Экосистема и сообщество:** готовые решения, наличие обучающих материалов;
- **Производительность:** возможность асинхронной обработки I/O-bound задач (парсинг, API запросы, взаимодействие с БД);
- **Взаимодействие с ML, CV и OCR:** наличие удобных библиотек для работы с моделями машинного обучения, компьютерного зрения и оптического распознавания текста.

На основе требований предметной области и сформулированных критериев выбора языка программирования выбран язык Python.

1.3.2 Обоснование выбора языка Python

Python обладает множеством преимуществ, которые делают его идеальным выбором для разработки серверной части исследуемой предметной области [8, 13, 2]:

1. **Интеграция с внешними API:** Наличие готовых библиотек для работы с CrossRef API и простая работа с JSON-объектами через стандартные структуры данных Python.
2. **Наличие мощных фреймворков:** Современные фреймворки для разработки веб-сервисов (FastAPI, Django, Flask) с различными подходами к архитектуре.
3. **Развитая экосистема для парсинга:** Готовые библиотеки для работы с PDF-документами, парсинга веб-сайтов и работы с API.
4. **Возможность параллельных вычислений:** Встроенные средства для асинхронной обработки I/O-bound задач и параллельных вычислений.
5. **Интеграция с ML, CV и OCR:** Готовые библиотеки для работы с моделями машинного обучения и компьютерного зрения.

1.3.3 Выбор веб-фреймворка

Ключевыми факторами выбора фреймворка для построения backend-части системы являются скорость разработки, поддержка асинхронности и масштабируемость продукта [7, 8, 3].

Среди имеющихся в Python фреймворков для построения backend-сервиса (см. табл. 1.3) был выбран FastAPI, основными преимуществами которого являются [7]:

1. **Асинхронность из "коробки":** Поддержка асинхронной обработки I/O-bound задач без использования дополнительных библиотек, что позволяет повысить производи-

Таблица 1.3 – Сравнение веб-фреймворков

Фреймворк	Описание	Преимущества	Недостатки
FastAPI	Быстрый и легкий в использовании для построения REST API	Быстрая разработка, валидация, типизация, ASGI, OpenAPI	Не имеет готовых ”батареек” по типу админ-панели, работы с пользователями.
Django	Монолитный фреймворк для построения веб-приложений	Встроенная ORM, Django-Admin, User Management, встроенная система безопасности	Нет полной поддержки асинхронности. Нет типизации. Нет валидации данных из ”коробки”.
Flask	Микрофреймворк для быстрой разработки RESTful API	Быстрая разработка, легкое масштабирование, хорошая документация	Не подходит для построения сложных API и веб-приложений

тельность сервиса при увеличении количества запросов.

2. **Автоматическая валидация данных:** Использование библиотеки Pydantic для автоматической валидации и преобразования данных, что позволяет избежать ошибок при обработке данных.
3. **Автоматическая генерация документации:** Генерация документации в формате OpenAPI для всех API-эндпоинтов.
4. **Инъекция зависимостей:** Поддержка паттерна Dependency Injection для обеспечения модульности и тестируемости кода.
5. **Гибкость при работе с БД:** Возможность использования любой ORM или библиотеки для работы с базами данных.

1.3.4 Выводы по разделу

Сравнительный анализ серверных технологий показал, что язык программирования Python в сочетании с фреймворком FastAPI является наиболее целесообразным выбором для реализации серверной части системы валидации научных публикаций. Python обладает развитой экосистемой библиотек для парсинга данных, работы с внешними API и асинхронной обработки запросов. FastAPI обеспечивает встроенную поддержку асинхронности, автоматическую валидацию данных и гибкость при выборе инструментов для работы с базами данных, что делает его оптимальным решением для построения масштабируемого REST API [3, 7].

1.4 Сравнительный анализ современных фронтенд-фреймворков

Аннотация. *Раздел содержит обзор и сравнительный анализ наиболее популярных фронтенд-фреймворков: Angular, Vue.js и React. На основе данных исследования State of JavaScript 2024 рассмотрены ключевые метрики (Awareness, Usage, Retention, Satisfaction), а также динамика популярности и удовлетворённости разработчиков. Проведён сравнительный анализ архитектуры, производительности, экосистемы и зрелости каждого инструмента. По результатам анализа обоснован выбор библиотеки React как наиболее рационального решения: она обеспечивает высокую производительность за счёт Virtual DOM, обладает крупнейшей экосистемой компонентов и поддерживается ведущими компаниями. React сочетает гибкость, зрелость и масштабируемость, что делает его оптимальным выбором для реализации интерактивного клиентского интерфейса.*

В процессе разработки веб-приложения особое значение имеет выбор технологического стека, определяющего архитектуру, производительность и устойчивость программного решения. Среди современных инструментов для построения интерфейсов были рассмотрены наиболее распространённые фреймворки и библиотеки: **Angular**, **Vue.js** и **React** [10, 2].

1.4.1 Критерии выбора фронтенд-фреймворка

По результатам анализа были сформулированы следующие критерии выбора фронтенд-фреймворка:

- **Архитектурная гибкость:** возможность адаптации структуры приложения под конкретные требования проекта;
- **Производительность:** высокая производительность и масштабируемость приложения;
- **Экосистема:** наличие большого количества библиотек и инструментов для разработки приложения;
- **Сложность освоения:** простота и скорость освоения фреймворка;

1.4.2 Сравненительный анализ современных фронтенд-фреймворков

Для объективного анализа и выбора фронтенд-фреймворка использованы данные ежегодного исследования **State of JavaScript** [10], которое проводится сообществом и является одним из наиболее авторитетных источников статистики по экосистеме JavaScript. Исследование включает опросы более 10,000 разработчиков ежегодно и анализирует тренды в ис-

пользовании, удовлетворенности и предпочтениях.

Ключевые метрики State of JavaScript:

- **Awareness:** уровень осведомленности разработчиков о фреймворке;
- **Usage:** уровень использования фреймворка;
- **Satisfaction:** уровень удовлетворенности разработчиков;
- **Interest:** процент тех, кто хочет его изучить;
- **Positivity:** процент тех, кто считает использование фреймворка положительным опытом;

Анализ данных State of JavaScript позволяет получить объективную картину о популярности и уровне удовлетворенности разработчиков различными фреймворками.

По результатам анализа данных State of JavaScript за 2024 год были получены следующие результаты:

React занимает лидирующую позицию на рынке фреймворков для фронтенд-разработки по большинству метрик:

- **Awareness:** 99% — практически все разработчики знают о React;
- **Usage:** 82% — используется в 82% проектов, что почти в 2 раза больше, чем у Vue.js и Angular (51% и 50% соответственно);
- **Satisfaction:** 74% — удовлетворенность разработчиков React находится на уровне Vue.js и Angular (79% и 50% соответственно);
- **Interest:** 37% — большинство разработчиков хотели бы изучить React или Vue.js (48%), но не Angular (17%);
- **Positivity:** 69% — большинство разработчиков считают использование React положительным опытом;

Сравнение Angular, Vue.js и React

На основе анализа таблицы 1.4 можно выделить следующие преимущества React над Angular и Vue.js применительно к построению фронтенд-части приложения:

- **Архитектурная гибкость:** Возможность использования различных библиотек и инструментов для построения приложения, что позволяет выбрать наиболее подходящее решение для конкретного проекта.
- **Производительность:** Использование виртуального DOM обеспечивает высокую производительность приложения и быструю ре-рендеризацию при частом обновлении данных.

Таблица 1.4 – Сравнение Angular, Vue.js и React

Критерий	Angular	Vue.js	React
Архитектурная гибкость	Строгая структура, ограниченная свобода выбора инструментов	Поддерживает постепенную интеграцию в существующие приложения и адаптивную модульность	Многократное использование компонентов, возможность использования различных библиотек
Производительность (рендеринг)	Более низкая из-за комплексной архитектуры и большого объёма встроенных модулей.	Эффективный виртуальный DOM, низкие затраты на рендеринг, быстрый отклик интерфейса.	Высокая производительность за счёт использования виртуального DOM
Экосистема	Включает в себя полный стек встроенных решений: маршрутизацию, формы, управление состоянием.	Уступает React по масштабу, но имеет большое количество официальных и сторонних модулей.	Наиболее развитая экосистема с множеством библиотек и инструментов
Сложность освоения	Наибольшая сложность: использование TypeScript, RxJS и концепции модульности	Наиболее проста для изучения: декларативный синтаксис шаблонов и логически разделённая структура файлов	Средняя сложность освоения: использование JSX и необходимость понимания принципов управления состоянием

– **Экосистема:** Наиболее развитая экосистема среди рассматриваемых фреймворков, что позволяет найти готовое решение для большинства задач.
– **Сложность освоения:** Средняя сложность освоения, что позволяет разработчику быстро освоить фреймворк.

1.4.3 Выводы по разделу

Проведённый сравнительный анализ современных фронтенд-фреймворков показал, что библиотека React является наиболее рациональным решением для построения фронтенд-части системы автоматической валидации научных публикаций. Это обусловлено оптимальным балансом между производительностью, гибкостью и зрелостью экосистемы. React обеспечивает модульный компонентный подход, высокую скорость рендеринга за счёт использования механизма Virtual DOM и широкую поддержку со стороны сообщества разработчиков и индустрии.

1.5 Выводы по главе 1

В результате проведённого исследования и сравнительного анализа существующих методов и технологий, направленных на автоматизацию валидации научных публикаций по официальному перечню рецензируемых изданий, были получены следующие результаты.

Анализ современного состояния системы документооборота показал, что существующая практика проверки публикаций вручную не отвечает требованиям эффективности и достоверности. Отсутствие открытого API, вариативность форматов PDF-документов и регулярные изменения в перечне ВАК создают предпосылки для ошибок и затрудняют интеграцию с информационными системами научных учреждений. Обоснована необходимость создания автоматизированной системы, обеспечивающей централизованное хранение, обновление и проверку данных с возможностью учёта исторических версий перечня.

В ходе анализа методов концептуализации предметной области установлено, что оптимальным подходом для моделирования данных является комбинированное использование ER-диаграмм для построения концептуальной модели базы данных и UML-диаграмм для описания семантических и поведенческих связей. Разработана трёхуровневая концептуальная модель, включающая слои версий перечня, сущностей и связей, что обеспечивает возможность отслеживания статуса журнала на любую дату и отражение временной динамики данных.

Сравнительный анализ серверных технологий показал, что язык программирования Python в сочетании с фреймворком FastAPI является наиболее целесообразным выбором для реализации серверной части системы. Данный технологический стек обеспечивает поддержку асинхронной обработки запросов, встроенную валидацию данных, широкие возможности интеграции с внешними API и высокую производительность при работе с I/O-bound задачами, включая парсинг PDF-документов и взаимодействие с CrossRef API.

В рамках анализа современных фронтенд-фреймворков установлено, что библиотека React обладает оптимальным балансом между производительностью, гибкостью и зрелостью эко системы. React обеспечивает модульный компонентный подход, высокую скорость ренде ринга за счёт использования механизма Virtual DOM и широкую поддержку со стороны сообщества разработчиков и индустрии. Выбор данной технологии гарантирует устойчивость и масштабируемость клиентской части веб-приложения.

Таким образом, в первой главе сформирована концептуальная основа проектирования системы: обоснована её актуальность, определены ключевые проблемы и требования, выбраны методы моделирования предметной области и определён технологический стек раз-

работки. Полученные результаты создают методологическую базу для реализации архитектуры и прототипирования автоматизированной системы валидации научных публикаций, что станет предметом рассмотрения в последующих главах.

1.6 Постановка задачи

Цель работы

Целью данной работы является разработка и реализация прототипа информационной системы автоматической валидации научных статей по официальному перечню рецензируемых научных изданий Высшей аттестационной комиссии (ВАК) Российской Федерации, обеспечивающей проверку соответствия публикации перечню с учётом даты публикации статьи и специальности автора. Основной задачей является устранение критической проблемы отсутствия единого программного интерфейса для проверки соответствия научных публикаций перечню ВАК, что в настоящее время требует ручной проверки на сайте портала и затрудняет интеграцию в информационные системы вузов и научных учреждений.

Задачи работы

Для достижения поставленной цели необходимо решить следующие задачи:

1. Построение концептуальной модели базы данных для качественной оценки предметной области;
2. Постановка сценариев выполнения задач в предметной области (проведение семантического анализа и формализации процессов);
3. Описание архитектуры серверной части системы, а также взаимодействия с внешними системами;
4. Проектирование и реализация базы данных с учётом концептуальной модели и сценариев выполнения задач;
5. Разработка RESTful API для взаимодействия с серверной частью системы;
6. Разработка фронтенд-части системы с использованием React;
7. Тестирование и отладка системы;
8. Документирование системы с полным описанием всех API-эндпоинтов и функциональности;

Ожидаемые результаты

Разработанный прототип информационной системы автоматической валидации научных статей по официальному перечню рецензируемых научных изданий Высшей аттестационной комиссии (ВАК) Российской Федерации должен обеспечивать:

- **Работающий прототип системы:** Полнофункциональная информационная система, готовая к использованию и ко вводу в опытную эксплуатацию, состоящая из парсера PDF, серверной части (REST API) и веб-интерфейса.
- **Реляционная СУБД**: Спроектированная и реализованная база данных с полной схемой, включающей все необходимые таблицы, индексы и ограничения целостности.
- **Парсер перечней:** Инструмент автоматического парсинга, обеспечивающий извлечение информации о журналах с точностью 99%, с учётом исторических версий перечня.
- **Веб-интерфейс:** Интерактивный и удобный веб-интерфейс, позволяющий пользователям легко взаимодействовать с системой, просматривать результаты валидации и управлять перечнями.
- **Интерфейс взаимодействия с серверной частью системы:** бэкенд-часть системы, позволяющая интегрировать и использовать её в других сервисах.
- **Тестирование и отладка:** Система должна содержать набор модульных (unit) и интеграционных (integration) тестов, а также бенчмарки для оценки производительности и масштабируемости системы.
- **Техническая документация:** Полная документация проекта, включающая описание архитектуры, документацию по использованию API, инструкции по развёртке, руководство разработчика.

2. Разработка моделей и алгоритмов автоматической валидации научных публикаций

В настоящей главе представлены результаты разработки ключевых моделей и алгоритмов, составляющих методологическую основу системы автоматической валидации. Описана формальная модель предметной области, включающая сущности «Журнал», «Специальность», «Версия перечня» и их семантические связи. Разработан алгоритм парсинга PDF-документов перечней ВАК, учитывающий вариативность форматов и обеспечивающий точность извлечения данных не менее 99%. Представлен метод валидации публикаций, основанный на сопоставлении даты публикации статьи с временными интервалами включения журнала в перечень на соответствующую дату. Описан алгоритм нормализации и дедупликации данных о журналах, поступающих из различных источников (PDF-перечни, CrossRef API). Разработана обобщенная архитектура системы, определяющая взаимодействие основных компонентов: парсера, базы данных, REST API и веб-интерфейса. Представлены интерфейсы модулей и протоколы обмена данными между компонентами системы.

2.1 Алгоритмы парсинга данных из PDF-документов

Аннотация. *Раздел посвящён разработке алгоритмов автоматического извлечения данных из PDF-документов перечней ВАК. Описаны подходы к парсингу табличной информации с учётом вариативности форматов PDF-документов. Представлены алгоритмы распознавания структуры таблиц, извлечения текстовых данных о журналах (ISSN, названия, специальности), обработки многостраничных документов и нормализации извлечённых данных. Описаны методы обработки различных форматов представления ISSN, распознавания специальностей и их кодов, извлечения временных интервалов включения журналов в перечень. Представлены алгоритмы валидации и очистки данных для обеспечения точности парсинга не менее 99%.*

2.2 Концептуальная модель базы данных

Аннотация. *Раздел описывает концептуальную модель базы данных для системы автоматической валидации научных публикаций. Представлена трёхуровневая архитектура данных, включающая слой версий перечня, слой журналов и специальностей, и слой связей между сущностями. Описаны основные сущности предметной области: Journal (Журнал), Specialty*

(Специальность), JournalVersion (Версия перечня), JournalSpeciality (Включение журнала в перечень), ArticleValidation (Валидация статьи). Представлена ER-диаграмма концептуальной модели с описанием атрибутов сущностей, первичных и внешних ключей, связей между сущностями. Описаны семантические ограничения целостности данных, обеспечивающие корректность хранения информации о версионности перечней и временных интервалах включения журналов.

2.3 Семантические связи между сущностями предметной области

Аннотация. *Раздел описывает семантические связи между сущностями предметной области и их представление в модели данных. Представлены UML-диаграммы классов, отражающие отношения между сущностями: ассоциации, агрегации и композиции. Описаны семантические связи между журналами и специальностями через сущность JournalSpeciality с указанием временных интервалов включения/исключения. Представлены связи между версиями перечня и журналами, обеспечивающие поддержку версионности данных. Описаны связи между статьями и результатами их валидации через сущность ArticleValidation. Представлены алгоритмы работы с семантическими связями для ответа на запросы о статусе журнала на конкретную дату и валидации публикаций с учётом временных аспектов данных.*

3. Проектирование системы автоматической валидации научных публикаций

В настоящей главе представлены детальные результаты проектирования системы автоматической валидации научных публикаций. Сформулированы функциональные требования, включающие возможности загрузки и парсинга перечней ВАК, валидации публикаций по ISSN и дате, поиска журналов по специальностям, просмотра истории изменений перечня. Разработаны диаграммы вариантов использования (use case), описывающие сценарии работы различных категорий пользователей. Представлена трёхуровневая архитектура системы (presentation layer, business logic layer, data layer) с детализацией компонентов каждого уровня и протоколов их взаимодействия. Спроектирована физическая модель базы данных, включающая схему таблиц с индексами, внешними ключами и триггерами для поддержки версионности и обеспечения целостности данных. Обоснован выбор технологического стека: Python 3.11+ с фреймворком FastAPI для реализации REST API, React для frontend-части сервиса, а также выбор СУБД PostgreSQL. Детально описаны проектные решения по организации модульной архитектуры backend с применением паттернов Dependency Injection, Repository и Service Layer, обеспечивающих тестируемость и поддерживаемость кода.

3.1 Функциональные и нефункциональные требования к системе

Аннотация. *Раздел содержит формализованные функциональные и нефункциональные требования к системе автоматической валидации научных публикаций. Представлены требования к загрузке и парсингу перечней ВАК, валидации публикаций по различным критериям, REST API эндпоинтам, а также требования к производительности, надёжности, масштабируемости, безопасности и удобству использования системы.*

3.1.1 Функциональные требования

Функциональные требования определяют конкретные возможности системы, которые должны быть реализованы для выполнения задач валидации научных публикаций.

Требования к загрузке и парсингу перечней ВАК

1. **FR-1.1: Загрузка PDF-документов перечня ВАК.** Система должна обеспечивать как возможность загрузки PDF-файлов через веб-интерфейс, так и автоматический пар-

синг перечней ВАК из официального источника в качестве background task (задачи, выполняемые в фоновом режиме).

2. **FR-1.2: Парсинг PDF-документов.** Система должна автоматически извлекать информацию о журналах из загруженных PDF-документов с точностью не менее 99%. Извлекаемые данные включают:
   - Номер журнала в перечне ВАК;
   - Название журнала;
   - ISSN журнала (основной и электронный);
   - Специальности, по которым журнал включён в перечень;
   - Даты включения/исключения конкретной специальности, связанной с журналом из перечня.
3. **FR-1.3: Версионность перечней.** Система должна поддерживать хранение множественных версий перечня ВАК с указанием даты ”по состоянию на” и временной метки загрузки.
4. **FR-1.4: Валидация загруженных данных.** Система должна проверять корректность извлечённых данных и сообщать об ошибках парсинга или несоответствии формата.

Требования к валидации публикаций

1. **FR-2.1: Валидация по ISSN.** Система должна проверять, был ли журнал с указанным ISSN включён в перечень ВАК.
2. **FR-2.2: Валидация по дате публикации.** Система должна проверять, был ли журнал включён в перечень ВАК на дату публикации статьи, учитывая временные интервалы включения/исключения журнала.
3. **FR-2.3: Валидация по специальности.** Система должна проверять, включён ли журнал в перечень по указанной специальности на дату публикации статьи.
4. **FR-2.4: Валидация по авторам.** Система должна проверять, соответствует ли запрашиваемый автор списку авторов, указанному в метаданных статьи из CrossRef API.
5. **FR-2.4: Полная валидация публикации.** Система должна выполнять комплексную проверку публикации по ISSN, дате публикации, авторам и специальности одновременно, возвращая детальный результат валидации с указанием причины несоответствия (если публикация не прошла проверку).

Требования к REST API

1. **FR-4.1: Эндпоинт валидации публикации.** Система должна предоставлять REST API эндпоинт `POST /api/validate` для валидации научной публикации.
2. **FR-4.2: Эндпоинт получения журналов.** Система должна предоставлять REST API эндпоинт `GET /api/journals` для получения списка журналов с поддержкой пагинации и фильтрации.
3. **FR-4.3: Эндпоинт поиска журналов.** Система должна предоставлять REST API эндпоинт `GET /api/journals/search` для поиска журналов по различным критериям.
4. **FR-4.4: Эндпоинт загрузки перечня.** Система должна предоставлять REST API эндпоинт `POST /api/journal-lists` для загрузки новых версий перечня ВАК.
5. **FR-4.5: Документация API.** Система должна автоматически генерировать документацию API в формате OpenAPI (Swagger), доступную по адресу `/docs`.
6. **FR-4.6: Формат данных.** Все API-эндпоинты должны использовать формат JSON для обмена данными.

3.1.2 Нефункциональные требования

Нефункциональные требования определяют качественные характеристики системы, которые не относятся к её функциональности.

Требования к производительности

1. **NFR-1.1: Время отклика API.** Время отклика API для валидации публикации не должно превышать 2 секунд при нормальной нагрузке.
2. **NFR-1.2: Пропускная способность.** Система должна обрабатывать не менее 100 запросов на валидацию в секунду при нормальной нагрузке.
3. **NFR-1.3: Время парсинга PDF.** Парсинг одного PDF-документа перечня ВАК не должен превышать 5 минут для документа размером до 1500 страниц.
4. **NFR-1.4: Время поиска.** Поиск журналов в базе данных должен выполняться не более чем за 1 секунду при количестве записей до 10,000 журналов.

Требования к надёжности

1. **NFR-2.1: Доступность системы.** Система должна обеспечивать доступность не менее 99% времени (uptime) в рабочее время.

2. **NFR-2.2: Обработка ошибок.** Система должна корректно обрабатывать все виды ошибок (ошибки парсинга, ошибки доступа к БД, ошибки внешних API) и возвращать понятные сообщения об ошибках пользователю.
3. **NFR-2.3: Целостность данных.** Система должна обеспечивать целостность данных в базе данных через использование транзакций, внешних ключей и ограничений.
4. **NFR-2.4: Резервное копирование.** Система должна обеспечивать возможность резервного копирования базы данных с частотой не реже одного раза в сутки.

Требования к масштабируемости

1. **NFR-3.1: Масштабируемость базы данных.** Архитектура базы данных должна поддерживать хранение данных о не менее 10,000 журналов и 50 версиях перечня ВАК.
2. **NFR-3.2: Горизонтальное масштабирование.** Архитектура backend-части системы должна поддерживать горизонтальное масштабирование (добавление новых экземпляров сервера) без изменения кода приложения.
3. **NFR-3.3: Масштабируемость фронтенда.** Frontend-часть системы должна корректно работать при количестве одновременно подключённых пользователей до 1,000.

Требования к безопасности

1. **NFR-4.1: Валидация входных данных.** Система должна валидировать все входные данные для предотвращения SQL-инъекций, XSS-атак и других видов уязвимостей.
2. **NFR-4.2: Ограничение размера файлов.** Система должна ограничивать размер загружаемых PDF-файлов до 50 МБ.
3. **NFR-4.3: Защита от злоупотреблений.** Система должна реализовывать механизмы защиты от злоупотреблений (rate limiting) для предотвращения перегрузки API.
4. **NFR-4.4: Безопасное хранение данных.** Все данные должны храниться в базе данных с использованием современных методов шифрования.

Требования к удобству использования

1. **NFR-5.1: Пользовательский интерфейс.** Веб-интерфейс системы должен быть интуитивно понятным и удобным для использования без необходимости обучения.
2. **NFR-5.2: Адаптивность интерфейса.** Веб-интерфейс должен корректно отображаться на различных устройствах (десктоп, планшет, мобильные устройства).

3. **NFR-5.3: Сообщения об ошибках.** Все сообщения об ошибках должны быть понятными и информативными для пользователя.
4. **NFR-5.4: Документация.** Система должна предоставлять понятную документацию по использованию API и веб-интерфейса.

3.1.3 Требования к интерфейсам

Пользовательский интерфейс

Веб-интерфейс системы должен предоставлять следующие возможности:

- Аутентификация и авторизация пользователей;
- Форма для ввода данных статьи (ISSN, название, автор, дата публикации, специальность);
- Отображение результата валидации с детальной информацией;
- Пагинированный список последних валидаций статей пользователя;
- Просмотр метаданных валидируемой статьи;
- Форма для загрузки новых версий перечня (для администраторов).

Интерфейс REST API

REST API должен предоставлять следующие основные эндпоинты:

- `POST /auth/login` — аутентификация пользователя;
- `POST /auth/register` — регистрация нового пользователя;
- `POST /validate` — валидация публикации;
- `GET /validations` — получение списка валидаций;
- `GET /validations/id` — получение информации о конкретной валидации;
- `POST /admin/import` — загрузка новой версии перечня;
- `GET /docs` — интерактивная документация API (Swagger UI/ReDoc).

3.2 Техническая реализация backend-части системы

Аннотация. *Раздел описывает техническую реализацию серверной части системы на основе Python и FastAPI. Представлено обоснование выбора технологий с детальным описанием используемых библиотек для парсинга PDF, работы с базой данных, интеграции с внешними API. Описаны ключевые возможности FastAPI: асинхронность, автоматическая валидация данных, инъекция зависимостей и гибкость при работе с базами данных с примерами кода реализации.*

3.2.1 Детальное обоснование выбора Python

Python выбран для реализации серверной части системы благодаря широкому набору специализированных библиотек и зрелой инфраструктуре для построения REST API [8, 13, 2]. Ниже представлены конкретные библиотеки и их применение в системе:

Интеграция с CrossRef API

Для работы с CrossRef API используются следующие библиотеки:

- **habanero, crossrefapi, crossref-commons:** официальные библиотеки для работы с CrossRef API;
- Стандартные структуры данных Python (dict, list) для простой работы с JSON-объектами;
- Библиотеки `aiohttp` и `httpx` для асинхронных HTTP-запросов.

Фреймворки для разработки веб-сервисов

Для реализации REST API применяется следующий стек:

- **FastAPI:** современный, асинхронный фреймворк для разработки RESTful API и микросервисов;
- **SQLAlchemy:** ORM для работы с реляционной базой данных PostgreSQL;
- **Alembic:** инструмент для миграций базы данных;
- **Pydantic:** библиотека для валидации данных и автоматической генерации схем.

Экосистема для парсинга данных

Для парсинга PDF-документов перечней ВАК используются следующие библиотеки:

- **pdfplumber:** библиотека для извлечения текста и таблиц из PDF;
- **pdfminer.six:** низкоуровневая библиотека для парсинга структуры PDF;
- **PyPDF2:** альтернативная библиотека для работы с PDF-документами.

Для парсинга веб-сайтов и работы с API:

- **BeautifulSoup:** библиотека для парсинга HTML;
- **Playwright:** инструмент для автоматизации браузера при необходимости парсинга динамического контента;
- **Scrapy:** фреймворк для веб-скрапинга.

Параллельные вычисления и асинхронность

Для обеспечения высокой производительности используются:

- **asyncio:** стандартная библиотека для асинхронного программирования;
- **Uvicorn, Hypercorn:** ASGI-серверы для запуска FastAPI-приложения;
- **WebSocket (socketio, websockets):** для двунаправленной связи между клиентом и сервером;
- **multiprocessing, threading:** для параллельной обработки данных при парсинге.

3.2.2 Детальная реализация FastAPI

Асинхронность из ”коробки”

FastAPI поддерживает асинхронную обработку запросов из ”коробки”, что критически важно для I/O-bound задач (запросы к API, взаимодействие с БД, парсинг файлов). Пример реализации асинхронного эндпоинта:

```python
from fastapi import FastAPI, Depends

app = FastAPI()

@app.post('/validate')
async def validate_article(
    article: Article,
    doi_service: Annotated[DoiService, Depends(get_doi_service)],
    validation_service: Annotated[
        ValidationService, Depends(get_validation_service)
    ],
    logging_service: Annotated[
        LoggingService, Depends(get_logging_service)
    ]
) -> ResponseSchema:
    article_data = await doi_service.fetch_article_data(article)
    validation_result = await validation_service.validate_article(
        article_data,
    )
    return validation_result
```

Автоматическая валидация данных

FastAPI использует библиотеку Pydantic для автоматической валидации и преобразования данных (JSON ↔ Python):

```python
"""Пример автоматической валидации данных"""

from fastapi import FastAPI
from pydantic import BaseModel, Field

class Article(BaseModel):
    """Модель валидации статьи."""
    title: str = Field(..., min_length=1, max_length=500)
    author: str = Field(..., min_length=1)
    issn: str = Field(..., pattern=r'^\d{4}-\d{4}$')
    publication_date: date

app = FastAPI()

@app.post('/validate')
async def validate_article(
    article: Article,
    validation_service: Annotated[
        AbstractValidationService, Depends(get_validation_service)
    ]
) -> ValidationResult:
    # Данные автоматически валидируются при парсинге запроса
    return await validation_service.validate(article)
```

FastAPI автоматически генерирует документацию в формате OpenAPI (Swagger), доступную по адресу `/docs`, где можно интерактивно тестировать все API-эндпоинты.

Инъекция зависимостей

FastAPI поддерживает паттерн Dependency Injection (DI) для обеспечения модульности и тестируемости кода [13, 7, 6]. Пример реализации DI с использованием абстрактных интерфейсов:

```python
from typing import Annotated, ABC, abstractmethod
from fastapi import FastAPI, Depends

class AbstractValidationService(ABC):  # Абстрактный интерфейс
    @abstractmethod
    async def validate(self, article: Article) -> ValidationResult:
        pass  # Реализация в подклассах

class FirstValidationService(AbstractValidationService):
    async def validate(self, article: Article) -> ValidationResult:
        # Логика валидации статьи

class SecondValidationService(AbstractValidationService):
    async def validate(self, article: Article) -> ValidationResult:
        # Другая логика валидации статьи

def get_validation_service(
    service_type: Literal["first", "second"] = "first"
) -> AbstractValidationService:
    if service_type == "first":
        return FirstValidationService()
    elif service_type == "second":
        return SecondValidationService()
    raise ValueError(f"Invalid service type: {service_type}")

@app.post('/api/validate')
async def validate_article(  # DI-контейнер
    article: Article,
    validation_service: Annotated[
        AbstractValidationService, Depends(get_validation_service)
    ]
) -> ValidationResult:
    return await validation_service.validate(article)
```

Гибкость при работе с базами данных

FastAPI не навязывает работу с конкретной ORM, что позволяет выбрать наиболее подходящее решение. В системе используется SQLAlchemy в сочетании с асинхронным драйвером `asyncpg` для PostgreSQL:

```
from sqlalchemy.ext.asyncio import AsyncSession, create_async_engine
from sqlalchemy.orm import sessionmaker

# Создание асинхронного движка БД
engine = create_async_engine(
    "postgresql+asyncpg://user:password@localhost/dbname",
    echo=True
)

# Создание сессии
AsyncSessionLocal = sessionmaker(
    engine, class_=AsyncSession, expire_on_commit=False
)

async def get_db() -> AsyncSession:
    async with AsyncSessionLocal() as session:
        yield session

@app.post('/journals')
async def create_journal(
    journal: JournalCreate,
    db: Annotated[AsyncSession, Depends(get_db)]
) -> JournalResponse:
    db_journal = Journal(**journal.model_dump())
    db.add(db_journal)
    await db.commit()
    await db.refresh(db_journal)
    return JournalResponse.model_validate(db_journal,
                                          from_attributes=True)
```

3.3 Техническая реализация frontend-части системы

Аннотация. *Раздел описывает техническую реализацию клиентской части системы на основе библиотеки React. Описана архитектура React-приложения: структура компонентов, управление состоянием, интеграция с Backend API через Axios. Приведены примеры реализации компонентов для валидации публикаций и отображения результатов с описанием их интеграции.*

3.3.1 Детальное обоснование выбора React

React выбран для реализации клиентской части системы на основе анализа метрик State of JavaScript 2024 [10]:

Метрики популярности

По данным State of JavaScript 2024, React занимает лидирующую позицию:

- **Awareness:** 99% — практически все разработчики знают о React;
- **Usage:** 82% — используется в 82% проектов;
- **Satisfaction:** 74% — высокий уровень удовлетворённости;
- **Interest:** 37% — высокий интерес к изучению;
- **Positivity:** 69% — большинство разработчиков считают использование React положительным опытом.

Преимущества React для системы валидации

1. **Компонентный подход:** Модульная архитектура позволяет создавать переиспользуемые компоненты для валидации публикаций, отображения журналов и управления перечнями.
2. **Virtual DOM:** Обеспечивает высокую производительность при частом обновлении интерфейса, что критично для системы с динамическим поиском и фильтрацией.
3. **TypeScript поддержка:** Возможность использования TypeScript для обеспечения типобезопасности.

3.3.2 Архитектура React-приложения

Структура компонентов

Приложение организовано по модульному принципу:

```tsx
// Компонент для валидации статьи
interface ArticleValidationProps {
  onSubmit: (article: Article) => Promise<ValidationResult>;
}

export const ArticleValidation: React.FC<ArticleValidationProps> = ({
    onSubmit
}) => {
  const [article, setArticle] = useState<Article>({
    title: '',
    author: '',
    issn: '',
    publicationDate: ''
  });
  const [result, setResult] = useState<ValidationResult | null>(null);
  const [loading, setLoading] = useState(false);
  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setLoading(true);
    try {
      const validationResult = await onSubmit(article);
      setResult(validationResult);
    } finally {setLoading(false);}
  };
  return (
    <form onSubmit={handleSubmit}>
      <button type="submit" disabled={loading}>
        {loading ? 'Проверка...' : 'Проверить'}
      </button>
      {result && <ValidationResultDisplay result={result} />}
    </form>
  );
};
```

Управление состоянием

Для управления состоянием приложения используется React Context API и библиотека Zustand:

```
import { create } from 'zustand';

interface ValidationStore {
  journals: Journal[];
  selectedJournal: Journal | null;
  setJournals: (journals: Journal[]) => void;
  setSelectedJournal: (journal: Journal | null) => void;
}

export const useValidationStore = create<ValidationStore>((set) => ({
  journals: [],
  selectedJournal: null,
  setJournals: (journals) => set({ journals }),
  setSelectedJournal: (journal) => set({ selectedJournal: journal }),
}));
```

Интеграция с Backend API

Для взаимодействия с REST API используется библиотека Axios:

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:8000/api',
  headers: {
    'Content-Type': 'application/json',
  },
  withCredentials: true,
});
```

```
class ValidationService {
    async validateArticle(article: Article): Promise<
    ValidationResult> {
        const response = await api.post<ValidationResult>(
            "/validate", article
        );
        return response.data;
    }
    async getValidations(): Promise<ValidationResult[]> {
        const response = await api.get<ValidationResult[]>(
            "/validations"
        );
        return response.data;
    }
    // Другие методы для работы с валидацией
}
```

Интеграция компонентов

Все компоненты интегрируются через главный компонент приложения:

```
export const App: React.FC = () => {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<ArticleValidation />} />
          <Route path="validations" element={<ValidationList />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
};
```

Такая архитектура обеспечивает:

- **Модульность:** каждый компонент отвечает за отдельную функциональность;
- **Переиспользуемость:** компоненты могут использоваться в разных частях приложения;
- **Тестируемость:** каждый компонент можно тестировать изолированно;
- **Масштабируемость:** легко добавлять новые компоненты и функциональность.

3.4 Архитектура приложения

Аннотация. *Раздел описывает архитектуру системы, основанную на принципах Domain Driven Design с применением паттерна Unit of Work. Представлены ключевые сущности предметной области и их взаимосвязи. Детально описан паттерн Unit of Work: его основные функции, интеграция с паттерном Repository, реализация на основе асинхронной сессии SQLAlchemy с примерами кода. Описаны преимущества применения Unit of Work и его интеграция с механизмом Dependency Injection FastAPI.*

3.4.1 Domain Driven Design в контексте системы валидации

Domain Driven Design представляет собой подход к разработке программного обеспечения, в котором основное внимание уделяется моделированию предметной области (domain) [5, 16]. В контексте системы автоматической валидации научных публикаций предметная область включает следующие ключевые сущности:

- **Journal (Журнал):** основная сущность, представляющая научное издание с его атрибутами (ISSN, название, номер журнала в перечне ВАК);
- **Specility (Научная специальность):** сущность, описывающая область науки, по которой журнал включён в перечень;
- **JournalVersion (Версия перечня):** сущность, представляющая версию официального перечня ВАК на определённую дату;
- **JournalSpeciality (Включение журнала в перечень):** сущность, связыющая журнал, специальности и временной интервал включения/исключения журнала в перечень.
- **ArticleValidation (Валидация статьи):** сущность, связывающая статью, журнал, результат и метаданные о валидации.

Применение DDD в системе валидации обеспечивает:

1. **Ясность бизнес-логики:** сложная логика валидации публикаций инкапсулируется в доменных сервисах и сущностях [5, 16], что делает код более понятным и поддержи-

ваемым;

2. **Изоляцию предметной области:** бизнес-логика отделена от технических деталей реализации (база данных, веб-фреймворк), что упрощает тестирование и модификацию;
3. **Универсальный язык:** использование терминов предметной области (ВАК, перечень, валидация, специальность) создаёт единый язык общения между разработчиками и экспертами предметной области.

3.4.2 Паттерн Unit of Work

Паттерн Unit of Work (Единица работы) является ключевым элементом архитектуры, основанной на принципах Domain Driven Design [9, 16]. Он служит для координации операций, выполняемых над объектами домена, обеспечивая их согласованность и целостность в рамках одной транзакции.

Основные функции паттерна Unit of Work

Паттерн Unit of Work выполняет следующие функции:

1. **Отслеживание изменений:** Unit of Work следит за состоянием объектов, загруженных из базы данных, фиксируя их изменения, добавления или удаления. Все операции над доменными объектами регистрируются в контексте единицы работы [9].
2. **Управление транзакциями:** Паттерн гарантирует, что все изменения, внесённые в объекты домена, будут сохранены в базе данных в рамках одной транзакции, обеспечивая атомарность и согласованность данных. При возникновении ошибки все изменения откатываются.
3. **Оптимизация производительности:** Объединяя несколько операций в одну транзакцию, Unit of Work снижает количество обращений к базе данных, что повышает эффективность работы приложения. Например, при валидации множества публикаций все изменения могут быть сохранены одним вызовом метода `commit()`.

Интеграция Unit of Work с паттерном Repository

В контексте Domain Driven Design, Unit of Work часто используется совместно с паттерном Repository [2]. Репозитории предоставляют абстракцию для доступа к данным, а Unit of Work управляет изменениями этих данных, обеспечивая их согласное сохранение. Такой подход способствует чёткому разделению ответственности и упрощает тестирование и сопровождение кода [4, 16, 9].

Реализация Unit of Work в системе валидации

В системе валидации научных публикаций паттерн Unit of Work реализован на основе асинхронной сессии SQLAlchemy. Ниже представлена структура реализации:

```python
from abc import ABC, abstractmethod
from typing import AsyncContextManager
from sqlalchemy.ext.asyncio import AsyncSession


class UnitOfWork(IUnitOfWork):
    """Реализация единицы работы с использованием AsyncSession."""

    def __init__(self, session_factory: AsyncSession):
        self._session_factory = session_factory
        self._session: AsyncSession | None = None
        self._repositories: dict[Type[IRepository], IRepository] = {}

    async def commit(self) -> None:
        if self._session is None:
            raise RuntimeError("Session is not initialized")
        await self._session.commit()

    async def rollback(self) -> None:
        if self._session is None:
            raise RuntimeError("Session is not initialized")
        await self._session.rollback()

    def get_repository(self, repository_type: Type[IRepository]):
        """Получение репозитория для использования в сервисах."""
        if repository_type not in self._repositories:
            self._repositories[repository_type] = repository_type(
                self._session,
            )
        return self._repositories[repository_type]
```

Применение Unit of Work в сервисном слое

Пример использования Unit of Work в сервисе валидации:

```python
class ValidationService:
    def __init__(self, uow: IUnitOfWork, doi_service: IDoiService):
        self._uow = uow
        self._doi_service = doi_service

    async def validate_and_log_article(
        self,
        article: Article
    ) -> ValidationResult:
        journal_repo = self._uow.get_repository(JournalRepository)
        journal = await journal_repo.get_by_issn(article.issn)
        if not journal:
            return ValidationResult(
                is_valid=False,
                reason="Журнал не найден"
            )

        validation_result = await self._validate_publication_date(
            journal, article.publication_date
        )
        # Логирование результата валидации в рамках той же транзакции
        logging_repo = self._uow.get_repository(LoggingRepository)
        await logging_repo.create_validation_log(
            article=article,
            result=validation_result,
            timestamp=datetime.now()
        )
        # Сохранение всех изменений атомарно
        await self._uow.commit()
        return validation_result
```

3.4.3 Преимущества применения Unit of Work в системе

Применение паттерна Unit of Work в архитектуре системы валидации научных публикаций обеспечивает следующие преимущества:

1. **Атомарность операций:** Все изменения, связанные с одной бизнес-операцией (например, валидация публикации и логирование результата), сохраняются в базе данных атомарно. При возникновении ошибки все изменения откатываются, что гарантирует целостность данных.
2. **Упрощение управления транзакциями:** Разработчику не требуется явно управлять открытием и закрытием транзакций для каждой операции. Unit of Work инкапсулирует эту логику, предоставляя простой интерфейс через методы `commit()` и `rollback()`.
3. **Повышение производительности:** Несколько операций с базой данных могут быть объединены в одну транзакцию, что уменьшает количество обращений к базе данных и улучшает общую производительность системы.
4. **Улучшение тестируемости:** Unit of Work может быть легко заменён на mock-реализацию в тестах, что позволяет изолированно тестировать бизнес-логику без реального доступа к базе данных.
5. **Согласованность данных:** Паттерн гарантирует, что все изменения, внесённые в рамках одной единицы работы, будут сохранены согласованно, что особенно важно для сложных операций, затрагивающих несколько сущностей домена [4].

3.4.4 Интеграция Unit of Work с Dependency Injection

В системе валидации Unit of Work интегрирован с механизмом Dependency Injection FastAPI, что обеспечивает прозрачное управление жизненным циклом транзакций:

```python
from fastapi import FastAPI, Depends
from typing import Annotated, AsyncContextManager

app = FastAPI()

async def get_uow() -> AsyncContextManager[IUnitOfWork]:
    """Dependency для получения Unit of Work."""
    return UnitOfWork(AsyncSessionLocal)

@app.post('/validate')
```

```python
async def validate_article(
    article: Article,
    uow: Annotated[AsyncContextManager[IUnitOfWork], Depends(get_uow)],
    validation_service: Annotated[
        ValidationService, Depends(get_validation_service)
    ]
) -> ValidationResult:
    async with uow:
        result = await validation_service.validate_and_log_article(
            article
        )
        await uow.commit()
        return result
```

Такая архитектура обеспечивает согласованное применение паттерна Unit of Work во всём приложении и упрощает управление транзакциями в каждом API-эндпоинте.

4. Программная реализация и экспериментальная проверка системы автоматической валидации научных публикаций

В настоящей главе представлены результаты практической реализации и экспериментальной проверки разработанной системы. Описан состав и структура реализованного программного обеспечения: backend-сервис на FastAPI, включающий модули парсинга PDF-документов, работы с базой данных через SQLAlchemy ORM, интеграции с CrossRef API и REST API эндпоинты; frontend-приложение на React с компонентами валидации публикаций, поиска журналов, просмотра истории перечней и административной панели; база данных PostgreSQL с реализованной схемой и системой миграций. Представлены основные сценарии работы пользователей с описанием интерфейсов: для исследователей — форма валидации публикации с получением мгновенного результата проверки; для администраторов — загрузка и обработка PDF-перечней, мониторинг системы; для внешних систем — интеграция через REST API с примерами запросов и ответов. Разработаны тестовые примеры, включающие модульные тесты для алгоритмов парсинга и валидации, интеграционные тесты для проверки взаимодействия компонентов.

4.1 Описание реализации

Аннотация. *Раздел содержит детальное описание практической реализации системы автоматической валидации научных публикаций. Представлена архитектура реализованного программного обеспечения с описанием компонентов backend-части на FastAPI: модули парсинга PDF-документов перечней ВАК, работа с базой данных через SQLAlchemy ORM с применением паттерна Unit of Work, интеграция с CrossRef API для получения метаданных статей, REST API эндпоинты для валидации публикаций и управления системой. Описана реализация frontend-приложения на React: компоненты для валидации публикаций, отображения результатов валидации, пагинированного списка валидаций пользователя, аутентификации и авторизации. Представлена структура базы данных PostgreSQL с описанием таблиц, индексов, внешних ключей и триггеров. Описаны основные сценарии работы пользователей с примерами интерфейсов и взаимодействия с системой. Приведены примеры кода ключевых компонентов системы с пояснениями реализации.*

4.2 Тестирование

Аннотация. *Раздел посвящён описанию процесса тестирования системы автоматической валидации научных публикаций. Представлена стратегия тестирования, включающая модульные тесты для проверки отдельных компонентов системы (алгоритмы парсинга PDF-документов, логика валидации публикаций, работа с базой данных), интеграционные тесты для проверки взаимодействия между компонентами (интеграция с CrossRef API, работа с базой данных через Unit of Work, взаимодействие между backend и frontend), тесты производительности для оценки времени отклика API и пропускной способности системы. Описаны используемые инструменты тестирования (pytest для Python, Jest для React), подходы к мокированию внешних зависимостей и созданию тестовых данных. Приведены результаты тестирования с показателями покрытия кода тестами, описанием выявленных ошибок и их исправления. Представлены примеры тестовых сценариев для критических функций системы.*

4.3 Развитие в будущем

Аннотация. *Раздел содержит описание направлений дальнейшего развития системы автоматической валидации научных публикаций. Рассмотрены потенциальные улучшения функциональности: расширение возможностей парсинга для работы с различными форматами перечней ВАК, добавление поддержки валидации публикаций по дополнительным критериям, интеграция с другими внешними API для получения метаданных публикаций, реализация автоматического обновления перечней ВАК через планировщик задач. Представлены предложения по улучшению производительности: оптимизация запросов к базе данных, кэширование часто используемых данных, масштабирование системы для обработки больших объёмов запросов. Описаны возможности расширения архитектуры: реализация микросервисной архитектуры, добавление очередей сообщений для асинхронной обработки задач, интеграция с системами мониторинга и логирования. Обсуждены перспективы интеграции системы с информационными системами вузов и научных учреждений.*

Список литературы

[1] *CrossRef API*. URL: https://www.crossref.org/documentation/retrieve-metadata/rest-api/ (дата обр. 19 окт. 2025).

[2] P. M. Debani, R. S. Surender и K. R. Kshirod. «Modern tools and current trends in web-development». В: *Indonesian Journal of Electrical Engineering and Computer Science* (). DOI: 10.11591/ijeecs.v24.i2.pp978-985.

[3] *FastAPI Best Practices*. 2025. URL: https://github.com/zhanymkanov/fastapi-best-practices (дата обр. 13 окт. 2025).

[4] B. L. Gorman. «Practical Entity Framework: Database Access for Enterprise Applications». В: (2020). DOI: https://link.springer.com/book/10.1007/978-1-4842-6044-9.

[5] V. Khononov. *Learning Domain-Driven Design*. O'Reilly Media, Inc., 2021. URL: https://www.oreilly.com/library/view/learning-domain-driven-design/9781098100124/.

[6] M. Laguna и др. *Business process modelling, simulation and design*. CRC Press, 2013. DOI: 10.1201/b14763.

[7] B. Lubanovic. *FastAPI. Modern Python Web Development*. O'Reilly Media, Inc., 2024. URL: https://www.oreilly.com/library/view/fastapi-modern/9781492090233/ (дата обр. 13 окт. 2025).

[8] B. Lubanovic. *Introducing Python*. O'Reilly Media, Inc., 2024. URL: https://www.oreilly.com/library/view/introducing-python-2nd/9781492051374/ (дата обр. 15 окт. 2025).

[9] *The Unit of Work Pattern and Persistent of Ignorance*. Microsoft, 2009. URL: https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/june/the-unit-of-work-pattern-and-persistence-ignorance (дата обр. 6 нояб. 2025).

[10] *State of JavaScript*. 2024. URL: https://stateofjs.com/en-US (дата обр. 18 окт. 2025).

[11] V. Wolfengagen, L. Ismailova и S. Kosikov. «Semantic neighborhood in cognitive modeling». В: *Cognitive Systems Research* (2025). DOI: 10.1016/j.cogsys.2025.101398.

[12] *Высшая аттестационная комиссия (ВАК) Российской Федерации*. URL: https://vak.gisnauka.ru/documents/editions (дата обр. 16 окт. 2025).

[13] Лавров Д. Н. и Лаврова Д. Д. «SOLID против GRASP». В: *Математические структуры и моделирование* (2025). DOI: 10.24147/2222-8772.2025.1.125-135.

[14] Г. М. Новикова и Т. В. Малютина. «Развитие инструментов моделирования предприятия для корпоративных систем управления». В: (2011). URL: https://cyberleninka.ru/article/n/the-development-of-enterprise-modeling-instrument-for-corporate-management-systems/viewer (дата обр. 15 окт. 2025).

[15] П. И. Чисников. «Unified Modeling Language (UML)». В: *Экономика, статистика и информатика. Вестник УМО* (2010). URL: https://cyberleninka.ru/article/n/unified-modeling-language-uml/viewer (дата обр. 15 окт. 2025).

[16] А. Швец. *Погружение в паттерны проектирования*. Refactoring.Guru, 2021. URL: https://refactoringu.ru/files/design-patterns-ru-demo.pdf.