Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Проектирование БД системы обогащения обучающей выборки для автоматизированной проверки отчета на соответствие нормативным требованиям»

Студент ИУ7-64Б (Группа) _____ (Подпись, дата) Разин А.В. (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) Строганов Ю. В. (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Во время обучения студентам регулярно приходится писать отчеты к различным видам работ (курсовые, лабораторные, научно-исследовательские работы и т. д.), при этом оформление работ должно соответствовать ГОСТ, что необходимо своевременно проверить и при необходимости отправить отчет на доработку, однако, количество студентов намного превышает количество нормоконтроллеров. Для ускорения процесса проверки возможно использование автоматических систем.

Целью курсовой работы является разработка базы данных обогащения обучающей выборки для автоматизированной проверки отчета на соответствие нормативным требованиям.

Для достижения цели научно-исследовательской работы требуется решить следующие задачи:

- проанализировать существующие решения;
- формализовать задачу и определить необходимый функционал;
- проанализировать способы хранения данных и системы управления базами данных, выбрать подходящую систему для поставленной цели;
- спроектировать базу данных, описать ее сущности и связи;
- разработать базу данных;
- исследовать зависимость времени выполнения запроса от числа получаемых запросов в секунду;
- сравнить скорость обработки запросов реализаций с кешем и без.

1 Аналитическая часть

В данной части работы будет описаны ошибки в отчетах, которые необходимо обнаружить, а также описаны участники процесса приема лабораторных работ, также будут описаны существующие средства автоматизации.

1.1 Основные ошибки в отчетах

Основные ошибки в отчетах описаны в приложении А.

1.2 Прием лабораторных работ

В не автоматизированной системе проверки отчетов на соответствие ГОСТ и дополнительным требованиям присутствуют две роли: студент, выполняющий некоторую работы, которая подразумевает написание отчета и нормоконтроллер, принимающий экспертное решение о соответствии предоставленного ему отчета необходимым требованиям.

1.2.1 Участники процесса приема работ

С помощью использования автоматической проверки отчета возможно сократить временные ресурсы, выделяемые нормоконтроллером на проверку отчетов студентов, однако, полностью отказаться от финального контроля результатов человеком невозможно, таким образом существует две роли при проверки отчета на соответствие ГОСТ, а именно: студент и нормоконтроллер.

1.2.2 Процесс приема работ

Студент отправляет отчет на проверку, а затем получает результат со списком ошибок (если имеются). Нормоконтроллер же анализирует отчет, составленный автоматической системой проверки, и при необходимости может внести необходимые правки. Диаграммы процесса проверки отчетов приведены на картинках 1.1–1.2, также приведена диаграмма BPMN 2.0, на которой представлено взаимодействие системы проверки отчетов, нормоконтроллера и студента (рисунок 1.3).

Использование автоматической проверки отчетов на соответствие ГОСТ и дополнительным требованиям сократит временные затраты на проверку отчетов.

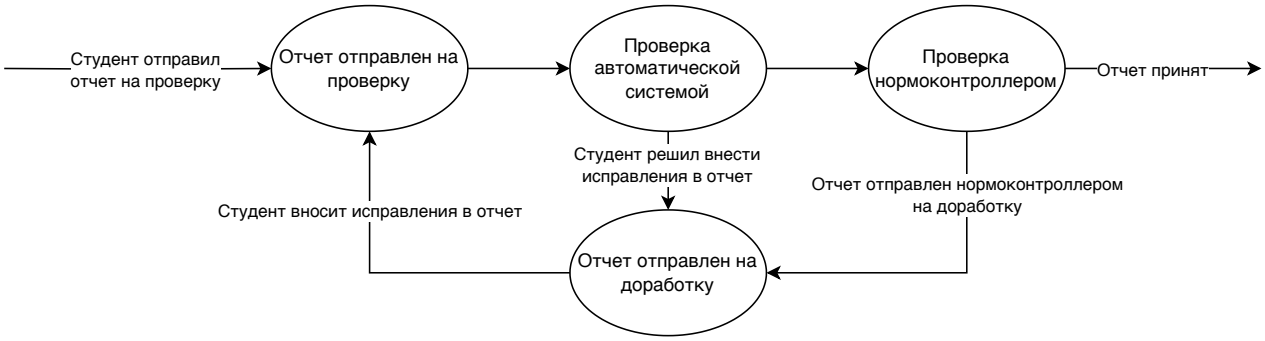

Рисунок 1.1 – Диаграмма состояний проверки отчета

Рисунок 1.2 – Диаграмма последовательности действий проверки отчета

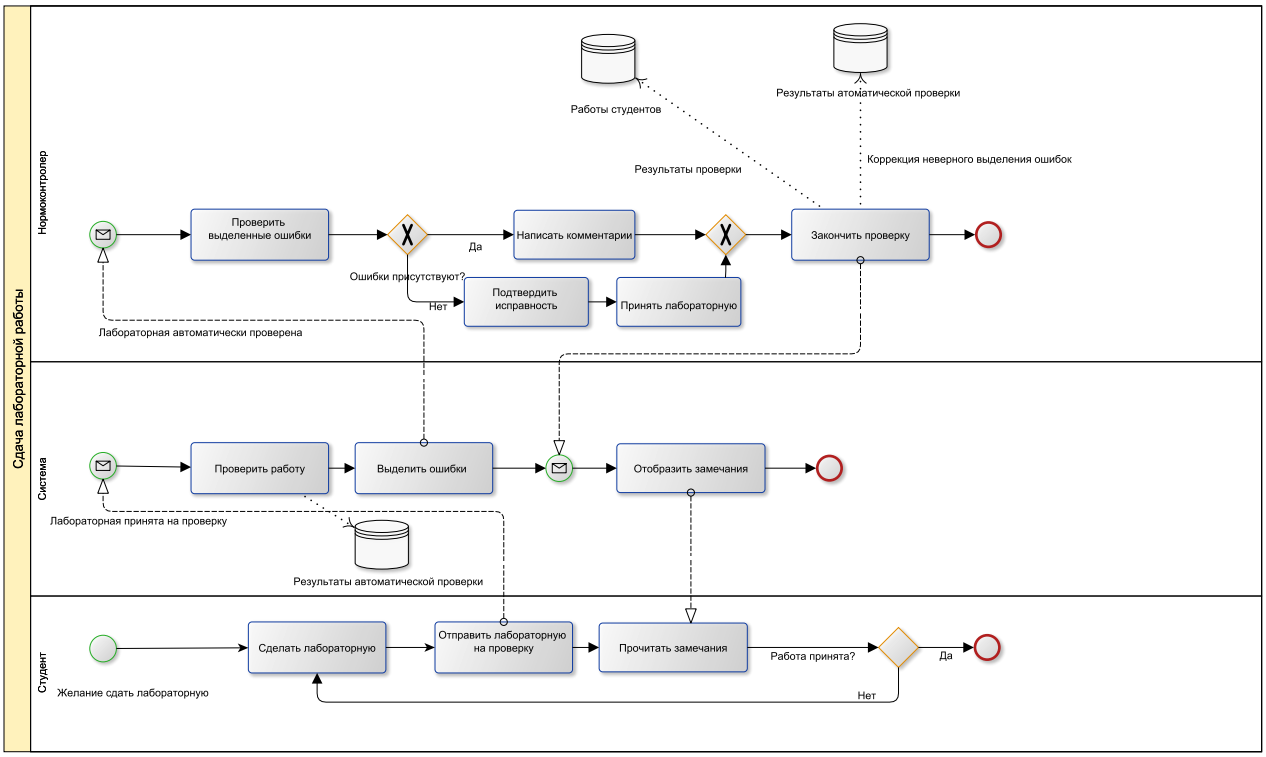

Рисунок 1.3 – BPMN 2.0 диаграмма сдачи лабораторной работы

1.3 Формализация требований к базе данных и приложению

В ходе выполнения курсовой работы необходимо разработать базу данных для хранения информации о студентах их работ, достижениях и результатов проверки работ студентов, также стоит хранить разметку основных частей отчетов студентов.

Необходимо спроектировать и разработать приложение, которое позволит проверять отчеты студентов, сохранять их, а также сохранять новые разметки основных частей отчета.

1.4 Анализ существующих средств автоматизации

Ввиду распространенности решаемой проблемы уже были созданы приложения для автоматизации проверки документов на соответствие стандартам.

Наиболее популярными из них являются:

1. ВКР-СМАРТ [1];
2. TestVkr [2];

3. Applitools visual testing [3];

Система ВКР-СМАРТ, предназначенная для проверки выпускных квалификационных работ (ВКР) студентов, представляет собой универсальную платформу, разработанную для системного хранения и проверки на заимствования ВКР и других работ обучающихся. Также система проверяет предложенные работы на выполнение всех требований ФГОС ВО и СПО, а также соответствие ГОСТам. После выполнения проверок, будет получен отчет о проценте заимствования и замечания по нарушенным стандартам [1].

Система ТЕСТ ВКР (Технический регламент проверки выпускных квалификационных работ) предназначена для проверки выпускных квалификационных работ студентов на объем заимствования и их размещения в электронно-библиотечной системе (ЭБС) университета. Система обеспечивает централизованное хранение и контроль за академическими работами студентов, а также их проверку на оригинальность и уникальность контента, также данную систему (без использования хранилища) возможно запускать локально, для проверки работы на нарушение ГОСТ [2].

Платформа Applitools позволяет использовать «визуальное тестирование» предназначенное для сравнения получаемого изображения с реальным. Этот метод особенно эффективен при выявлении ошибок во внешнем виде страницы или экрана, которые могут остаться незамеченными при традиционном функциональном тестировании. С использованием Applitools Eyes разработчики могут легко интегрировать визуальные тесты, которые могут быть использованы для выявления отклонений от стандартов в PDF [3].

Таблица 1.1 – Сравнение существующих средств автоматизации

Критерий	**ВКР СМАРТ**	**TestVkr**	**Applitools**
Проверка текстов	да	да	нет
Проверка элементов отчета	нет	нет	да
Наличие общего хранилища работ	да	да	нет
Возможность запуска локально	нет	*да	*да

В таблице 1.1 под «элементами отчета» подразумеваются таблицы, рисунки, схема алгоритмов, формулы, использование символа *, означает, что в этом случае проверка плагиата не производится.

1.5 Формализация информации, подлежащей хранению в проектируемой базе данных

Разрабатываемая база данных, должна содержать информацию о следующих сущностях:

- студент;
- нормоконтроллер;
- отчет студента;
- тип фрагмента отчета;
- фрагмент отчета;
- комментарий студента;
- достижение студента.

Сведения о каждой категории данных содержится в таблице 1.2.

Таблица 1.2 – Категории и сведения о данных

Категория	**Сведения**
Студент	ID студента, никнейм, имя, фамилия, дата регистрации, число сданных лабораторных
Отчет	ID отчета, номер попытки сдачи, файл отчета, время последнего обновления отчета, число проверок отчета, ID студента, значение того что отчет сдан
Тип фрагмента отчета	ID фрагмента, описание типа фрагмента, ID контроллера, создавшего фрагмент отчета
Выделенный фрагмент отчета	ID фрагмента отчета, изображение страницы отчета, данные фрагмента отчета (разметка), значение была ли ошибка проверена разметчиком, ID отчета, ID типа фрагмента отчета,ID типа задетектированного основного элемента отчета, ID проверяющего отчет
Комментарий	ID ошибки, на которую создается комментарий, ID комментария, данные комментария, ID студента, создавшего комментарий
Достижение	ID достижения, ID контроллера, создавшего достижение, данные достижения, описание достижения

На основе описанной информации была получена диаграмма сущность—связь, представленная на рисунке 1.4.

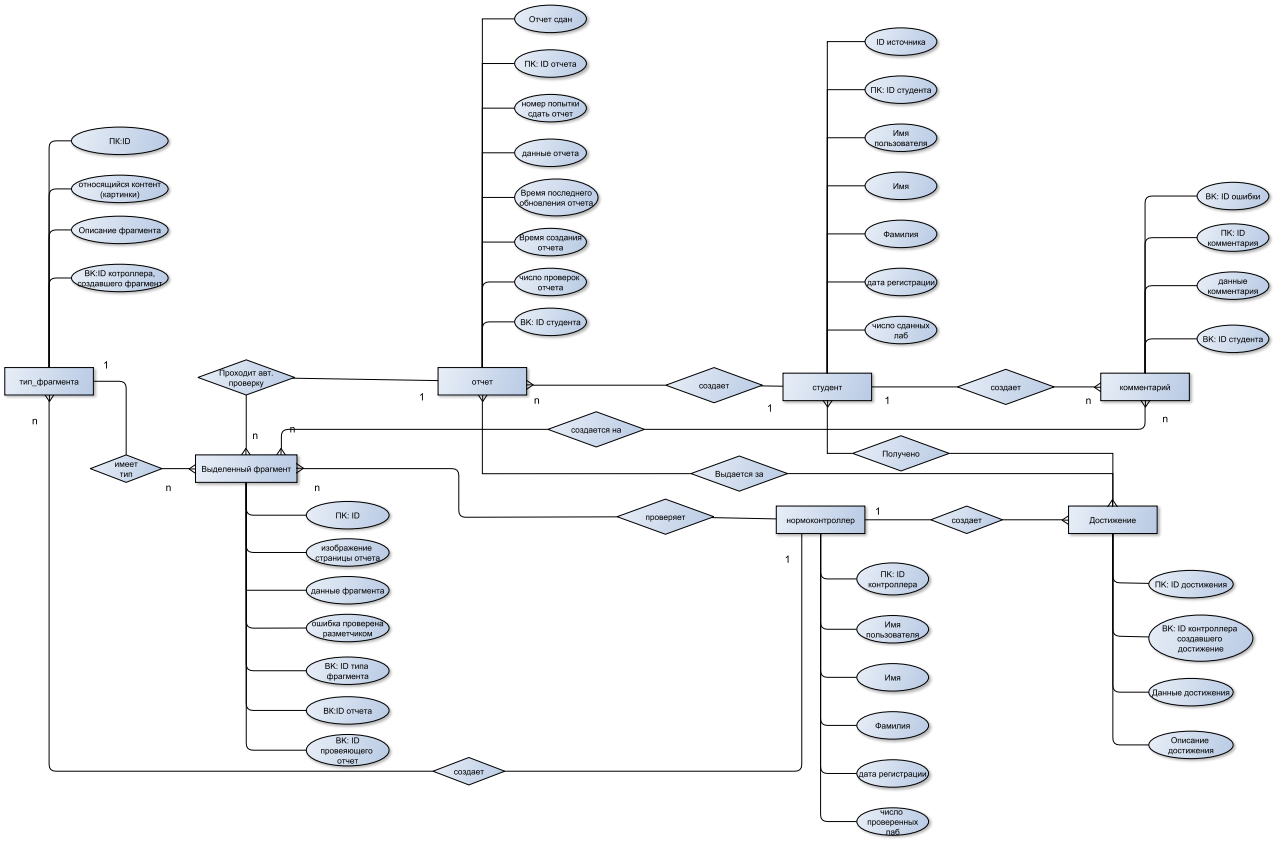

Рисунок 1.4 – Диаграмма сущность—связь

1.6 Формализация пользователей

При анализе предметной области были выделены следующие роли в системе:

1. Гость — данная роль представляет не авторизованного пользователя, соответственно он может авторизоваться и войти в систему.

2. Студент — данная роль позволяет создавать комментарии к отчетам, просматривать отправленные документы и ошибки в них.

3. Нормоконтроллер — данная роль позволяет создавать типы разметок, разметки элементов отчета, просматривать отправленные документы и ошибки в них, а также создавать, удалять, модифицировать достижения и создавать комментарии.

4. Админ — возможности данной роли аналогичны роли нормоконтроллера, однако данная роль также позволяет удалять типы элементов отчета и изменять роль пользователей (со студента на нормоконтроллера и наборот).

Для описания сценариев взаимодействия пользователей с системой была создана диаграмма прецедентов, представленная на рисунке 1.5.

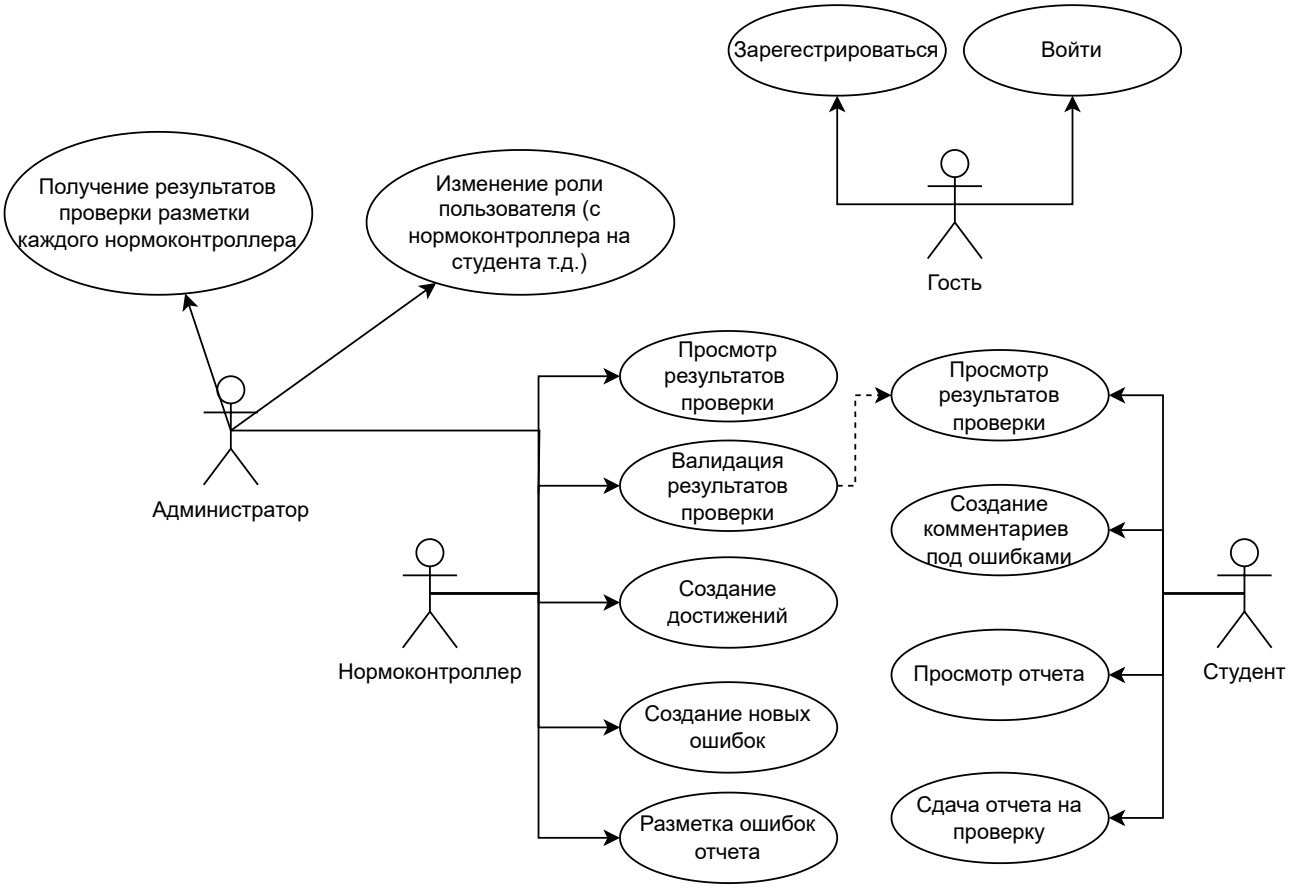

Рисунок 1.5 – Диаграмма прецедентов

1.7 Анализ существующих баз данных

База данных — это некоторый набор перманентных (постоянно хранимых) данных, используемых прикладными программными системами какого-либо предприятия [4].

Между физической базой данных (т.е. данными, которые реально хранятся на компьютере) и пользователями системы располагается уровень программного обеспечения, который можно называть по-разному: диспетчер базы данных (database manager), сервер базы данных (database server) или, что более привычно, система управления базами данных, СУБД (DataBase Management System — DBMS). Основная задача СУБД — дать пользователю базы данных возможность работать с ней, не вникая во все подробности работы на уровне аппаратного обеспечения [4].

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих

абстрактную машину доступа к данным, с которой взаимодействует пользователь [4]. Рассмотрим классификацию баз данных по различным моделям данных:

1. дореляционные;
2. реляционные;
3. постреляционные.

1.7.1 Дореляционные модели

Дореляционные системы можно разделить на три большие категории:

1. системы с инвертированными списками;
2. иерархические;
3. сетевые [4].

Иерархическая база данных представляется множеством деревьев: в вершинах дерева помещаются записи, состоящие из поименованных полей и представляющие экземпляры некоторого объекта предметной области. Записи связаны строго иерархическими отношениями — у записи-«потомка» не должно быть более одной записи-«предка» [5].

Сетевая модель данных представляет собой логическую модель данных, которая расширяет иерархический подход. В иерархических структурах каждая запись-потомок имеет ровно одного предка, в то время как в сетевой структуре данных потомок может иметь несколько предков [5].

Инвертированный список в общем случае — это двухуровневая индексная структура. На первом уровне находится файл или часть файла, в которой упорядоченно расположены значения вторичных ключей. Каждая запись с вторичным ключом имеет ссылку на номер первого блока в цепочке блоков, содержащих номера записей с данным значением вторичного ключа. На втором уровне находится цепочка блоков, содержащих номера записей, содержащих одно и то же значение вторичного ключа. При этом блоки второго уровня упорядочены по значениям вторичного ключа, на третьем уровне находится основной файл [6].

Механизм доступа к записям по вторичному ключу при подобной организации записей весьма прост. На первом шаге происходит поиск в области первого уровня заданное значение вторичного ключа, а затем по ссылке считываются блоки второго уровня, содержащие номера записей с заданным значением вторичного ключа, а далее уже прямым доступом загружается в рабочую область пользователя содержимое всех записей, содержащих заданное значение вторичного ключа [6].

1.7.2 Реляционные модели

Реляционная база данных — это такая база данных, которая воспринимается ее пользователями как множество переменных (т.е. переменных отношения), значениями которых являются отношения или, менее формально, таблицы [4].

Реляционная система, поддерживает реляционные базы данных и осуществляет операции над ними, включая RESTRICT (также известную как выборка или англ. SELECT), проекцию (англ. PROJECT) и соединение (англ. JOIN). Эти и подобные операции реляционной алгебры выполняются на уровне множеств [4].

1.7.3 Постреляционные модели

Современный (постреляционный) этап развития связан с использованием объектно-ориентированных технологий разработки программных систем и созданием СУБД нового поколения, унаследовавших все лучшее от дореляционных и реляционных систем. Постреляционные СУБД поддерживают объектные и объектно-реляционные модели данных и обеспечивают разработчикам возможность использовать объектно-ориентированные языки программирования, что дает таким системам технологические преимущества по сравнению с реляционными СУБД [5].

1.7.4 Хранение временных рядов

Отдельно стоит отметить набирающие популярность СУБД временных рядов. Временные ряды (англ. Time series) — это данные, претерпевающие некоторые изменения с течением времени, и фиксируемые в конкретные промежутки времени. Данные СУБД оптимизированы под частую запись данных

и скорость получения доступа к данным не настолько сильно зависит от числа хранимых данных, что характерно для реляционных баз данных, скорость работы которых уменьшается ввиду необходимости индексирования новых элементов. Однако во временных СУБД отсутствует механизм изменения записанных значений, так как считается что записанные метрики (данные) являются фактом в прошлом [7].

Вывод

В данном разделе были описаны основные ошибки студентов в отчетах по лабораторным работам, были выделены участники процесса приема лабораторных работ, формализован процесс приема лабораторных работ, а также рассмотрены существующие средства автоматизации проверки работ на соответствие стандартам.

2 Конструкторская часть

В данной части работы будет описана реализация базы данных, описана ER диаграмма БД и принципиальная схема БД, также будут описаны схемы триггеров и хранимые процедуры.

2.1 Формализация взаимодействия с приложением

Системе автоматической проверки необходимо проверить на правильность составные части отчета, что подразумевает детекцию рисунков, графиков, схем алгоритмов, списка используемых источников, а также формул.

На рисунках 2.1–2.2 представлены результаты разработки системы детекции составных частей.

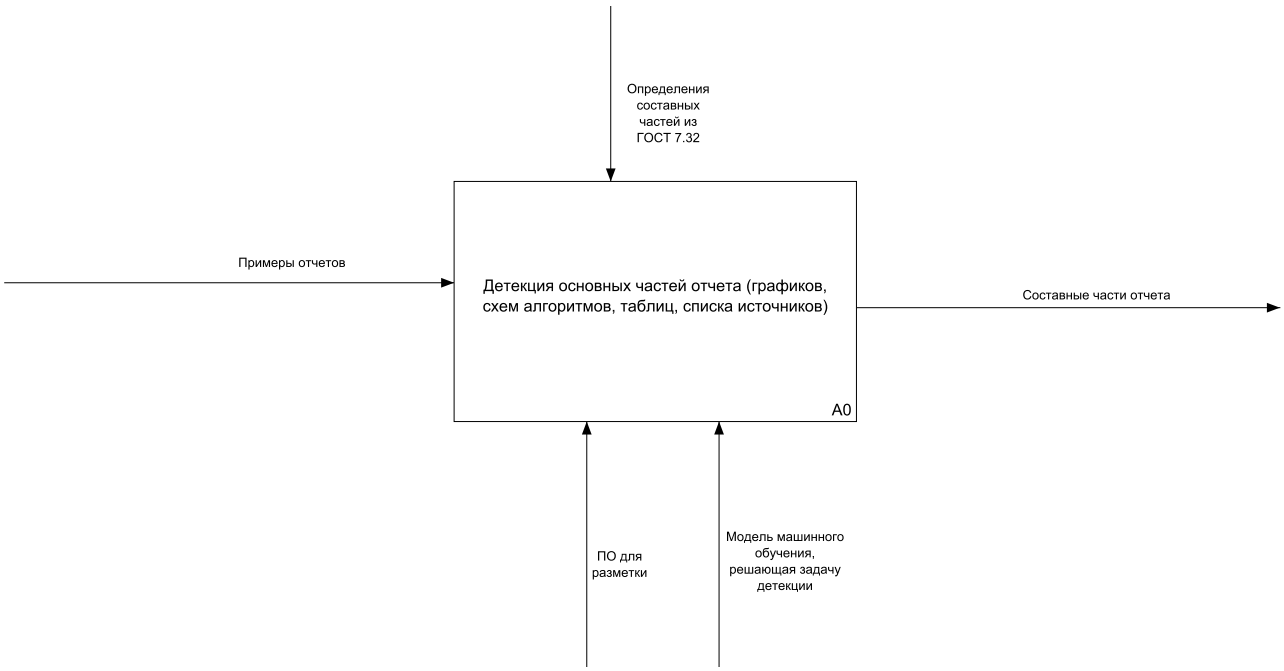

Рисунок 2.1 – IDEF0 обнаружение составных частей отчет 0 уровня

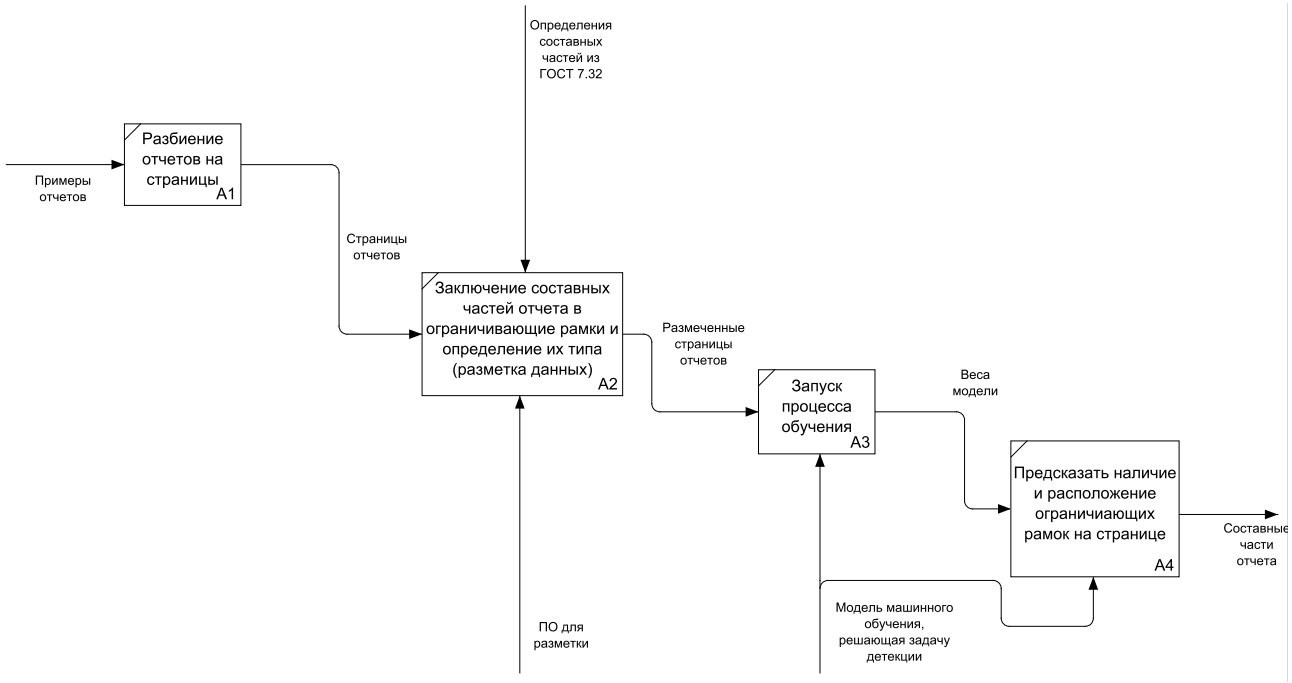

Рисунок 2.2 – IDEF0 обнаружение составных частей отчета 1 уровня

После решения задачи детекции необходимо проверить составные части на соответствие ГОСТ 7.32, однако, финальный вердикт должен выноситься экспертом. На рисунках 2.3–2.4 представлены результаты разработки данной системы.

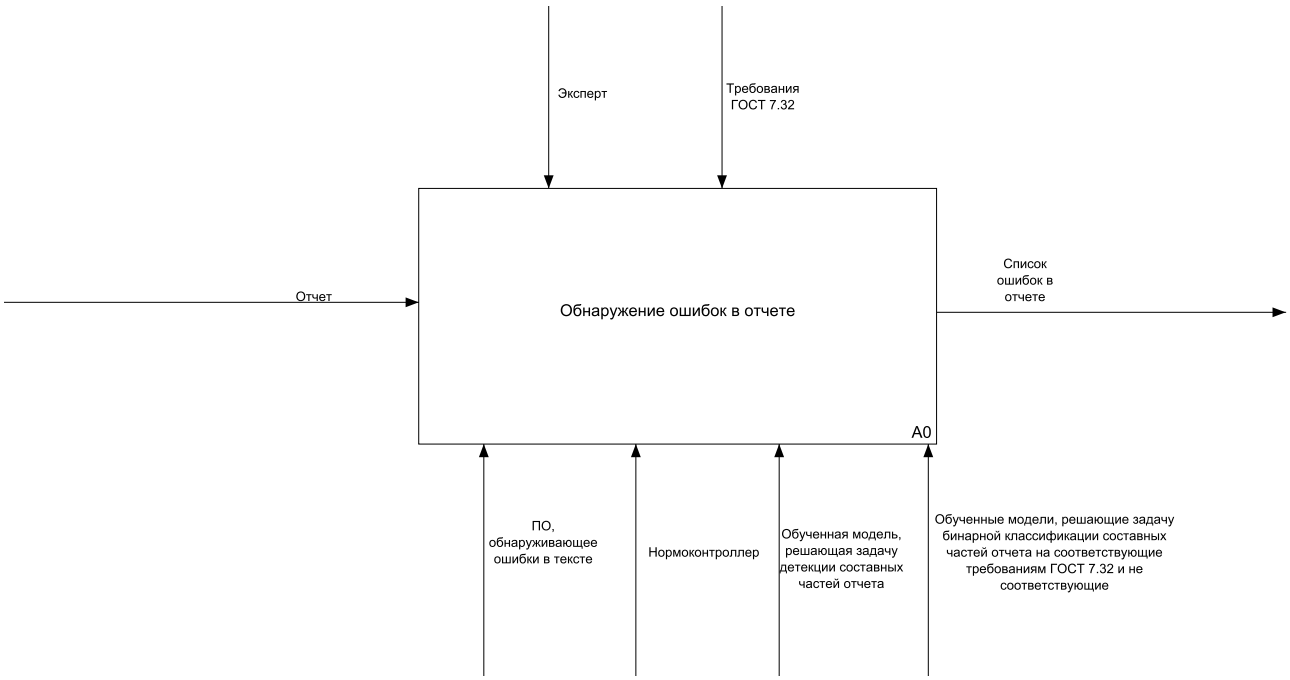

Рисунок 2.3 – IDEF0 проверки составных частей отчета на соответствие ГОСТ 7.32 0 уровня

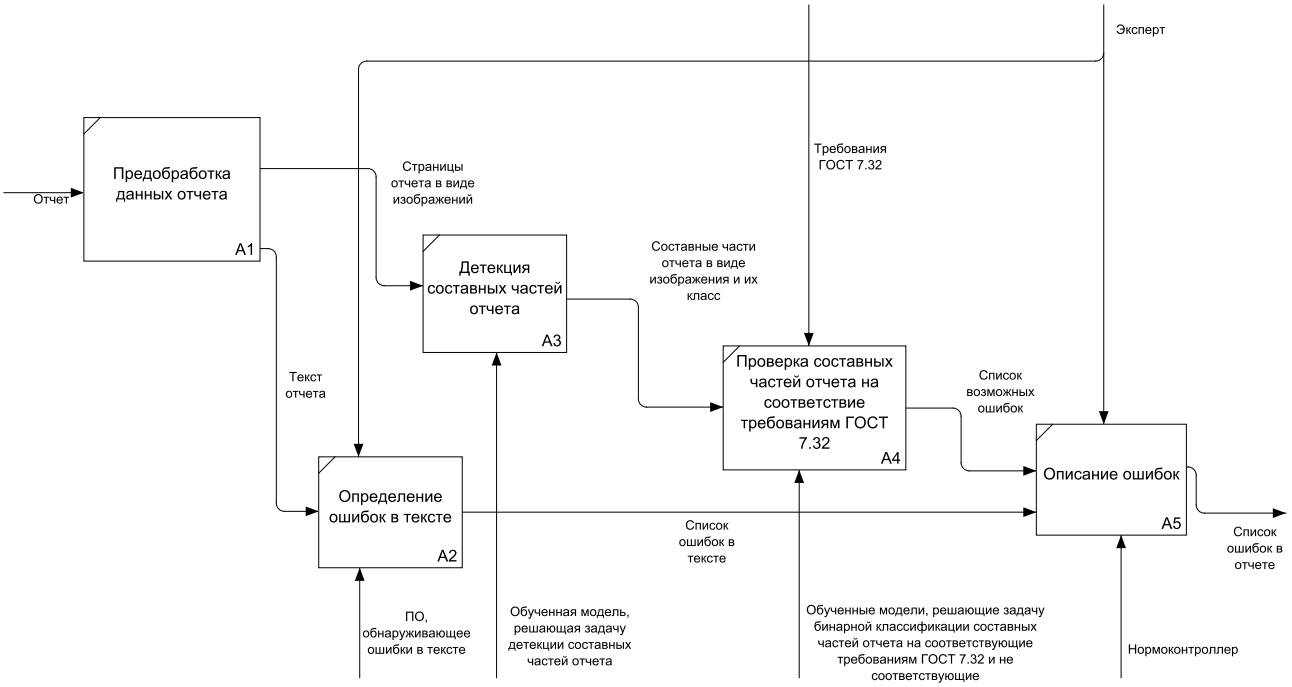

Рисунок 2.4 – IDEF0 проверки составных частей отчета на соответствие ГОСТ 7.32 1 уровня

2.2 Формализация работы системы проверки отчета

На рисунке 2.5, представлена схема BPMN 2.0 работы системы в процессе проверки отчета, а также всех взаимодействующих с ней акторов.

2.3 Формализация сущностей базы данных

В соответствие с диаграммами 2.3–2.4 были выделены следующие таблицы:

1. студент — сущность студента (имеет уникальный идентификатор для связи со сданными отчетами и комментариями);

2. нормоконтроллеер — сущность нормоконтроллера (имеет уникальный идентификатор для связи с проверенными отчетами и созданными ошибками);

3. выделенный фрагмент — сущность ошибок в отчетах студентов (хранит страницу отчета с ошибкой, а также данные, указывающие на местоположение ошибки, а также 2 типа фрагмента: тип ошибки и класс части отчета полученного в результате детекции, также хранит метаданные о проверке данной разметки преподавателем и валидности данной разметки в случае, если исправить детекцию невозможно);

4. тип фрагмента — сущность типов ошибок (необходима для введения новых типов ошибок, хранит описание ошибки);

5. отчет — сущность отчета студента (хранит все страницы попытки сдачи отчета, а также время сдачи и номер попытки сдачи);

6. комментарий — сущность комментария студента о ошибке (в случае ошибки преподавателя можно указать на это, кроме данных комментария хранит идентификатор студента, создавшего комментарий);

7. достижение — сущность награды для выдачи студентам, преуспевающим в выполнении лабораторных работ (хранит идентификатор студента, которому выдали награду, а также идентификатор отчета, за который было получено достижение).

2.4 Ограничения целостности базы данных

В данной части работы будут описаны описаны поля сущностей баз данных и их ограничения, для обеспечения целостности данных системы.

Таблица 2.1 – Ограничение целостности сущности документа

Поле	Тип	Ограничение
id	целое число	первичный ключ
page_count	целое число	не пустое значение
document_name	строка	не пустое значение
checks_count	целое число	не пустое значение
creator_id	целое число	не пустое значение
creation_time	дата и время	не пустое значение
has_passed	булевое значение	не пустое значение

Таблица 2.2 – Ограничение целостности сущности типа фрагмента

Поле	Тип	Ограничение
id	целое число	первичный ключ
description	строка	не пустое значение
creator_id	целое число	не пустое значение
class_name	строка	не пустое значение
is_valid	булевый тип	не пустое значение

Ограничение «первичный ключ» подразумевает то что поле сущности является первичным ключом данного отношения в реляционной модели данных.

Таблица 2.3 – Ограничение целостности сущности фрагмента

Поле	Тип	Ограничение
id	целое число	первичный ключ
page_data	байты изображения	не пустое значение
error_bb	массив из 4 вещественных чисел	не пустое значение
class_label	целое число	не пустое значение
creator_id	целое число	не пустое значение
type_label	целое число	не пустое значение
document_id	тип uuid	не пустое значение
was_checked	булевый тип данных	не пустое значение
is_valid	булевый тип данных	не пустое значение

Таблица 2.4 – Ограничение целостности сущности пользователя

Поле	Тип	Ограничение
id	целое число	первичный ключ
login	строка	уникальный,не пустое значение
password	строка	не пустое значение
name	строка	не пустое значение
surname	строка	не пустое значение
role_id	целое число	не пустое значение
role_type	строка	не пустое значение
group	строка	не пустое значение

Таблица 2.5 – Ограничение целостности сущности достижения

Поле	Тип	Ограничение
id	целое число	первичный ключ
picture	байты изображения	не пустое значение
text	строка	не пустое значение
creator_id	целое число	не пустое значение
granted_to_id	целое число	не пустое значение

Таблица 2.6 – Ограничение целостности сущности комментария

Поле	Тип	Ограничение
id	целое число	первичный ключ
description	строка	не пустое значение
creator_id	целое число	не пустое значение
report_id	целое число	не пустое значение

2.5 Ролевая модель

Была определена следующая ролевая модель:

1. Разметчик — имеет права доступа SELECT к таблице документов и INSERT к таблице выделенных фрагментов.

2. Добавляющий — имеет права доступа INSERT к таблице студентов и нормоконтроллеров

3. Пользователь — имеет права доступа INSERT к таблице комментариев, имеет права доступа SELECT и INSERT к таблице выделенных фрагментов.

4. Нормоконтроллер — является разметчиков, также имеет права доступа INSERT и SELECT к таблице тип_ошибки и таблице достижений.

5. Администратор имеет все права доступа ко всем таблицам.

6. Также выделяется работник очереди задача которого — работать с отчетами из очереди и помечать их как проверенные,имеет права доступа SELECT и UPDATE к очереди документов.

7. При проверке документов из очереди система работает в роли работника очереди, в иных случаях система использует роль администратора.

2.6 Определение модели базы данных

Необходимо определить модель базы данных, которая будет использована для хранения информации в системе. Реляционная модель баз данных позволяет хранить сущности в целостном виде и поддерживает сложные запросы, поэтому она является наилучшим выбором для поставленной задачи. Также необходимо хранить файлы журнала приложения, ввиду частого его пополнения и необходимости частой записи стоит использовать базу данных временных рядов.

2.7 Используемые триггеры

В базе данных присутствует триггер, запускающий процесс обнаружения ошибок в отчете при добавлении отчета в базу данных. При вставке экземпляра отчета в базу данных, в случае, если данный отчет не был до этого проверен и не имеет в имени файла VIP (такие файлы обслуживаются вне очереди), он будет добавлен в очередь отчетов. Очередь отчетов представляет собой таблицу, в который хранится ID отчета и его статус (не обработан, обработан, обработан с ошибкой).

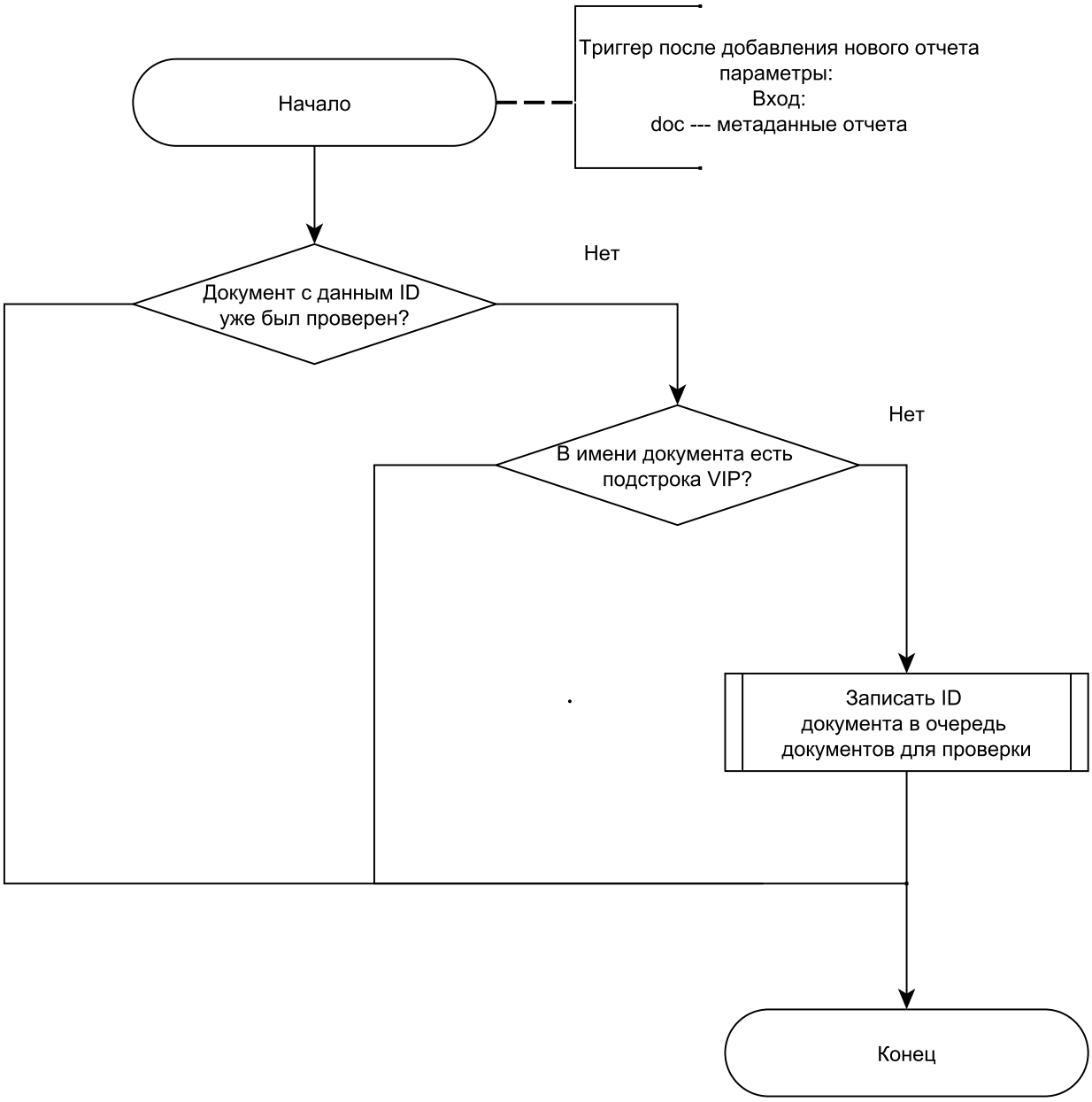

Рисунок 2.6 – Схема алгоритма триггера после добавления в таблицу метаданных отчетов

2.8 Используемые функции

Для извлечения метаданных документов с различными статусами готовности (0 — документ из очереди не был обработан, 1 — документ был успешно обработан, 2 — документ не был обработан связи с ошибкой) была разработана функция `getTasksReady`. Под метаданными в данном случае имеется в виду статус и идентификатор документа в очереди.

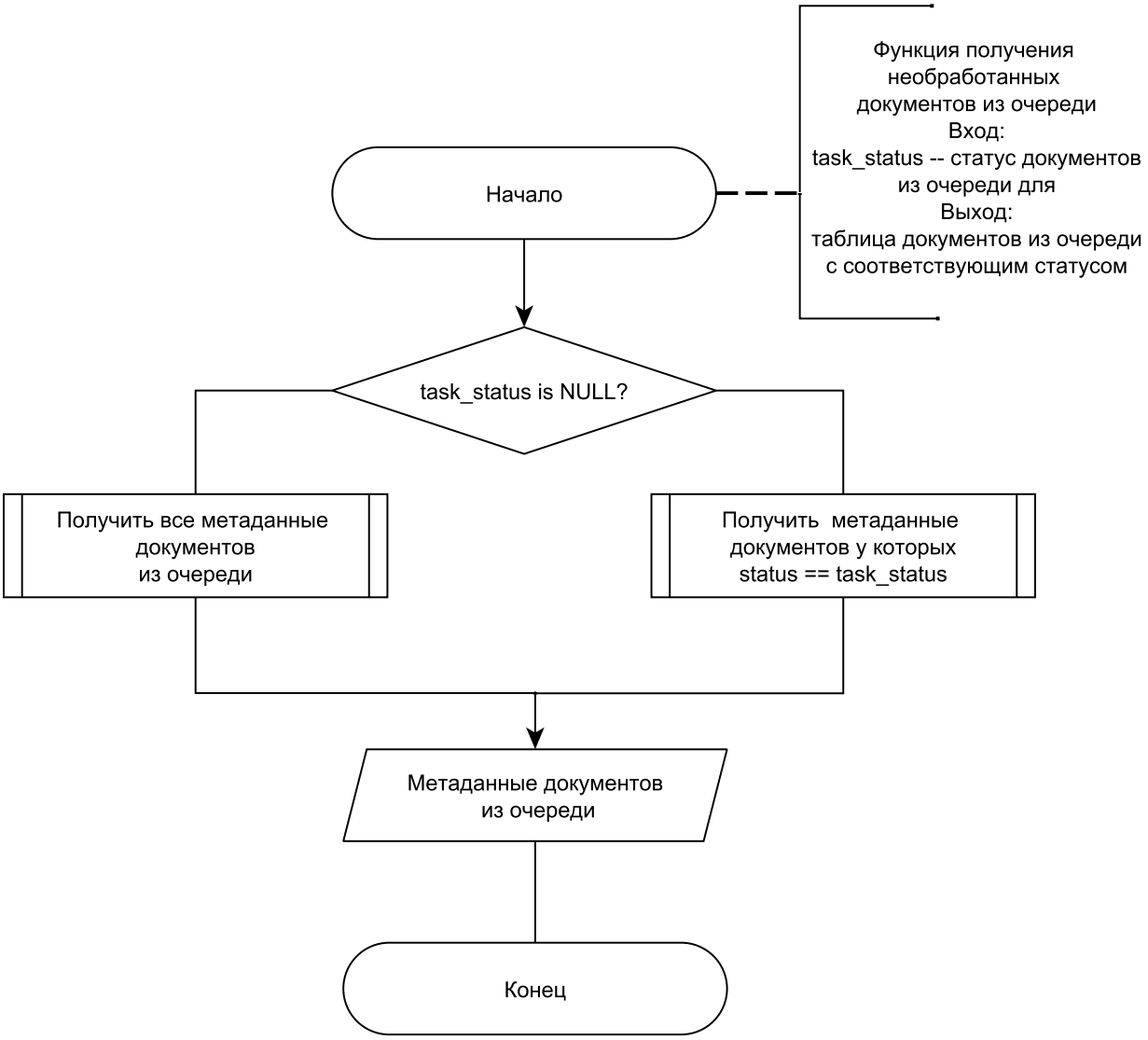

Рисунок 2.7 – Функция получения элементов из очереди с соответствующим статусом

Вывод

В данной части были формализованы процессы автоматической проверки, а также выделены различные виды пользователей: нормоконоктроллер, студент, система и администратор, после чего были определены права для каждого из пользователей.

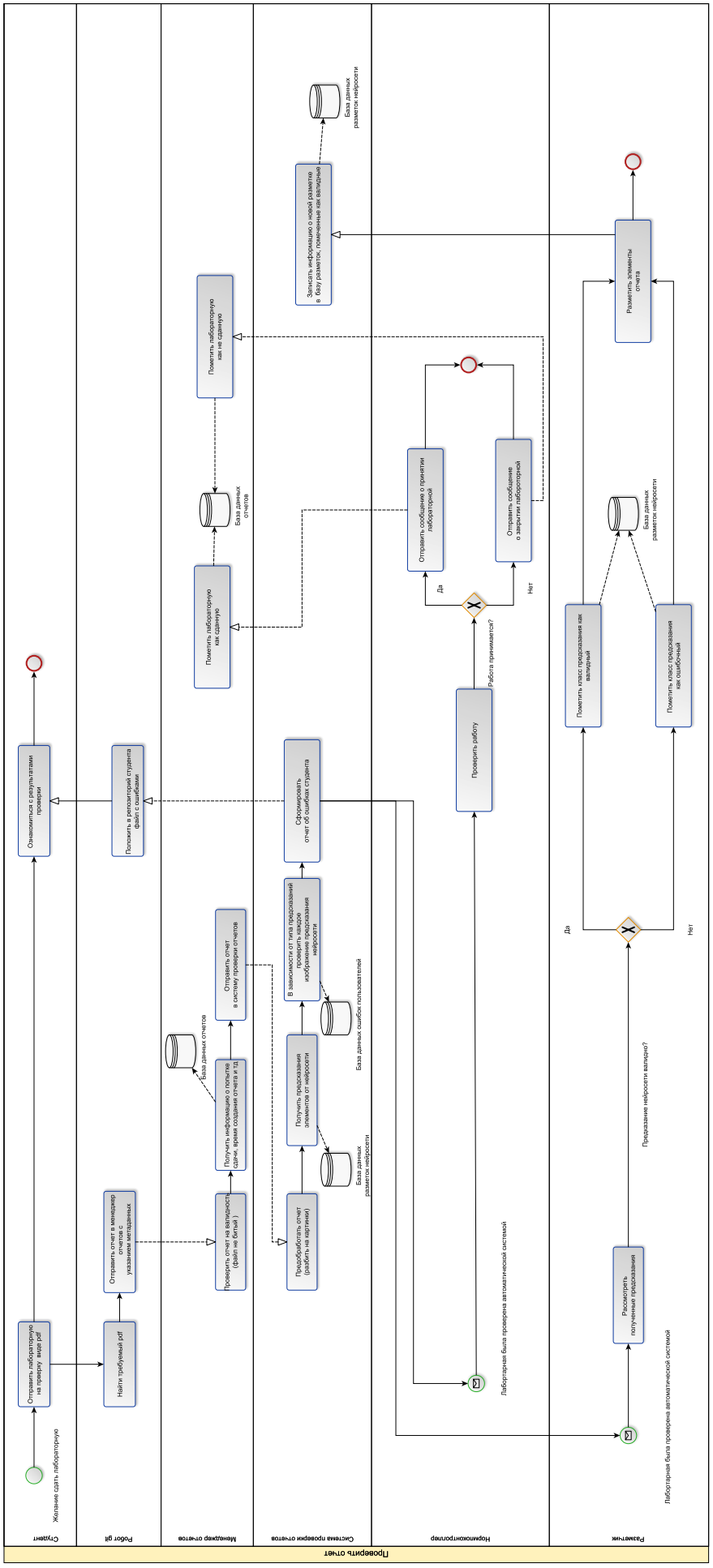

Рисунок 2.5 – Схема BPMN 2.0 работы системы

3 Технологическая часть

В данной части рассматривается выбор средств реализации, описывается реализация алгоритмов и построенных схем.

3.1 Выбор СУБД

В соответствии с приведенной диаграммой сущность—связь необходимо организовать хранение следующих типов данных.

1. Файлы отчетов и результатов проверки работ студентов.
2. Выделенные фрагменты и их описание.
3. Информацию о студентах, достижениях, комментариях.
4. Хранение файлов журнала системы (англ. logs) [8].

Также в выбранной СУБД должна быть возможность создания ролей и выдача прав в соответствии с ними. В конструкторском разделе была определена реляционная модель баз данных для хранения информации в системе, файлы отчетов и результатов проверки будут хранится в файловой системе сервера, мета информация для их получения будет также хранится в базе данных системы.

Рассмотрим наиболее популярные реляционные СУБД [9]:

1. PostgreSQL [10; 11].
2. MySQL [12].
3. MS SQL [13].

Сравнение будет произведено по следующим критериям:

1. Доступность в лицензии Российской Федерации.
2. Популярность среди разработчиков.
3. Наличие ролевой модели.

3.1.1 Доступность лицензии

PostgreSQL является является ПО с открытым исходным кодом, ввиду чего существуют несколько версий данного продукта [10]. На базе PostgreSQL была разработана СУБД Postgres Pro Certified,которая сертифицирована ФСТЭК РФ [11].Сертификация позволяет использовать ее для хранения информации:

1. В значимых объектах критической информационной инфраструктуры 1 категории, в государственных информационных системах 1 класса защищенности;

2. В автоматизированных системах управления производственными и технологическими процессами 1 класса защищенности;

3. В информационных системах персональных данных при необходимости обеспечения 1 уровня защищенности персональных данных;

4. В информационных системах общего пользования II класса.

MS SQL (Microsoft SQL Server) является продуктом компании Microsoft,в 2022 году ФСТЭК принял решение об отзыве сертификатов данной компании [14].

MySQL не является сертифицированном продуктом на территории Российской Федерации.

Все 3 СУБД имеют ролевую модель данных[10; 15; 16]. Стоит отметить что наибольший процент использования СУБД имеет MS SQL, затем идет PostgreSQl, затем MySQL [9].

В результате анализа полученных данных, была получена таблица 3.1. В таблице 3.1 числа в строке «Частота практического использования» обозна-

Таблица 3.1 – Сравнение наиболее популярных СУБД

СУБД / Критерии сравнения	PostgreSQL	MySQL	MS SQL
Уровень защищенности	1	нет	нет
Частота практического использования	2	3	1
Наличие ролевой модели	да	да	да

чают порядковый номер СУБД с точки зрения практического внедрения или использования [9].

Таким образом для решения поставленной задачи стоит выбрать СУБД PostgreSQL, так как занимает второе место по популярности и имеет сертификацию 1-го уровня защищенности(см. таблицу 3.1).

Также необходимо реализовать хранения файлов журнала приложения, для этого было решено использовать СУБД временных рядов. Так как InfluxDB поддерживает использование InfluxQL подобного SQL, а также является ПО с открытым исходным кодом стоит использовать ее для хранения файлов журнала [7].

3.2 Выбор средств реализации

Для реализации описанного функционала был выбран язык программирования GO, в силу следующих причин:

1. Существует библиотека для данного языка, позволяющая реализовывать запросы к базам данных на уровне языка [17].

2. Язык поддерживает создание интерфейсов и структур, что позволяет выделить основные объекты и сущности при разработке и использовать их при реализации [18].

3. GO имеет обширную документацию, в которой в каждом описываемом модуле (пакете) приведены примеры реализации и использования объектов модуля [19].

Для реализации нейронной сети был выбран язык Python, так как онн поддерживает фреймворк PyTorch, использование которого достаточно для решения задачи детекции основных частей отчета [20].

3.3 Схемы базы данных

Схема реализованной базы данных представлена на рисунке 3.1, PK — обозначают первичные ключи базы данных, FK — вторичные.

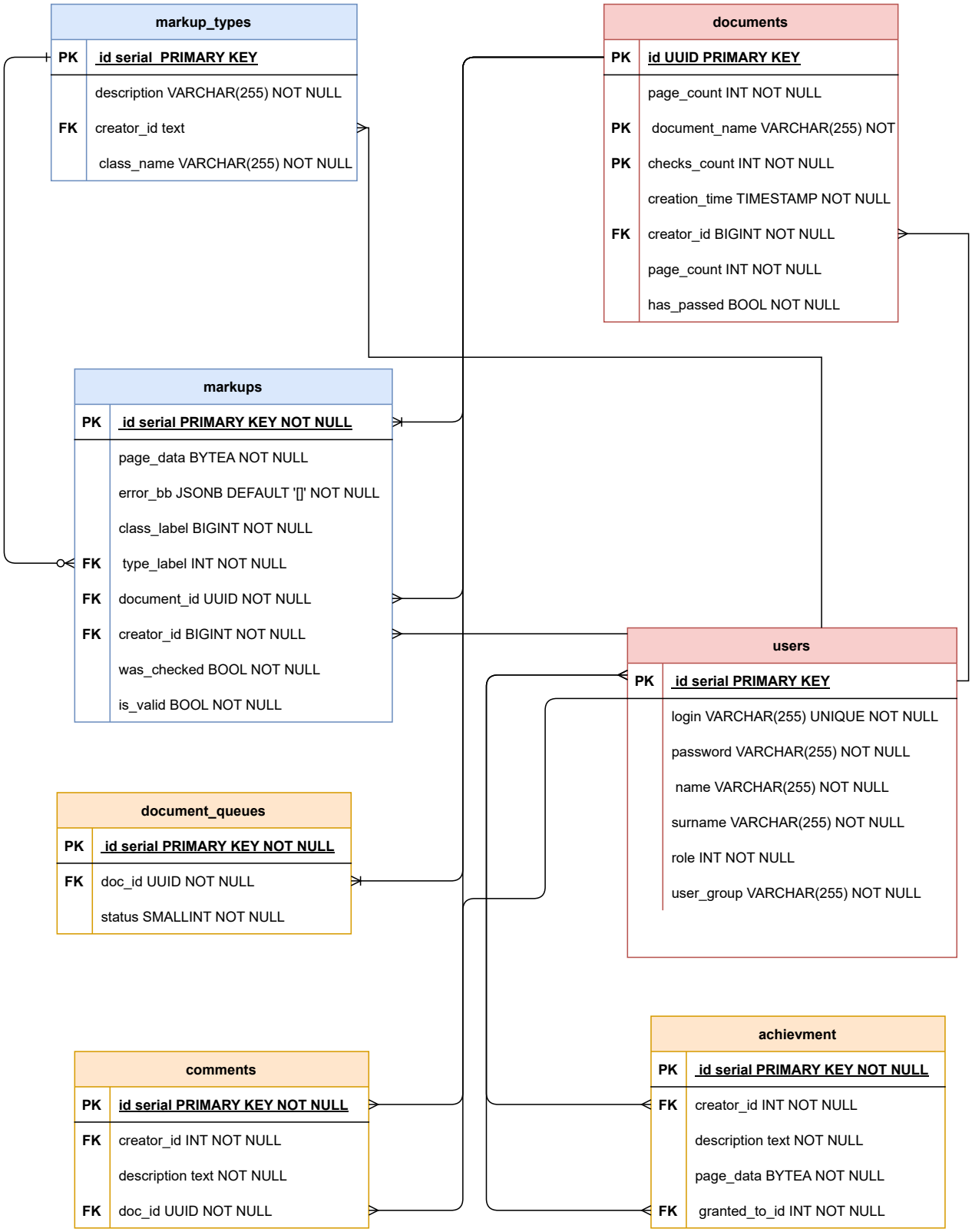

Рисунок 3.1 – Схема алгоритма триггера после добавления в таблицу метаданных отчетов

3.4 Реализация создания отношений

В листинге В.2 приведена реализация создания отношений. Отношения обозначают соответственно:

1. documents — сущность документа;
2. markup_types — сущность типов фрагмента отчета;
3. markups — сущность выделенного фрагмента;
4. users — сущность пользователя (контроллера, админа или студента);
5. document_queues — сущность элемента очереди отчета для проверки;
6. comments — сущность комментариев к отчету;
7. achievments — сущность достижения.

3.5 Реализация функций

В листинге В.3 приведены реализации спроектированных функций. Функция `getTasksReady` получает все документы из очереди, которые готовы к проверке и не были проверены до этого.

3.6 Реализация триггеров

В листинге В.4 приведены реализации спроектированных триггеров. Триггер `addToQueue`, при добавлении метаданных документа, в случае если в имени файла нет подстрока VIP и он не был уже проверен добавляет документ в очередь для проверки.

3.7 Тестирование разработанных функций базы данных

В данной части работы будет произведено тестирование разработанных функций и триггеров базы данных.

Рассмотрим следующие варианты использования разработанной процедуры `getTasksReady`:

1. Получение всех записей в таблице при передачи значения `NULL`.

2. Получение записей в таблице с соответствующим статусом (в данной системе статусы принимают значения 0–2, где 0 —- работа пользователя не проверена, 1 — работа пользователя проверена, 2 — ошибка в работе системы при проверке).

Состояние исходного отношения для тестирования представлено в таблице 3.2:

Таблица 3.2 – Исходное состояние таблицы для тестирования

ID	**doc_id**	**status**
1	710eb9fc-1118-4a66-837e-8e3d2d027a68	0
2	710eb9fc-1118-4a66-837e-8e3d2d027a67	0
3	710eb9fc-1118-4a66-837e-8e3d2d027a65	0
4	710eb9fc-1118-4a66-837e-8e3d2d027a61	0
5	5d92a123-b3a8-4f6d-b449-dc07913f60be	0
7	df6286ba-9993-4a15-928b-2ef0652dd699	1
9	5d92a123-b3a8-4f6d-b449-dc07913f60be	2

Тест был успешно пройден.

3.7.1 Получение всех значений очереди

Запрос вида `select * from getTasksReady(null);`.

Ожидание: будут получены все значения из исходного отношения.

Результат приведен в таблице 3.3.

Таблица 3.3 – Результаты получения всех значений из исходного отношения

ID	**doc_id**	**status**
1	710eb9fc-1118-4a66-837e-8e3d2d027a68	0
2	710eb9fc-1118-4a66-837e-8e3d2d027a67	0
3	710eb9fc-1118-4a66-837e-8e3d2d027a65	0
4	710eb9fc-1118-4a66-837e-8e3d2d027a61	0
5	5d92a123-b3a8-4f6d-b449-dc07913f60be	0
7	df6286ba-9993-4a15-928b-2ef0652dd699	1
9	5d92a123-b3a8-4f6d-b449-dc07913f60be	2

3.7.2 Получение значений очереди в соответствии со статусом

Запрос вида `select * from getTasksReady(1::int2);`.

Ожидание: будут получены элементы отношения со статусом равным 1.

Результат приведен в таблице 3.4. Тест был успешно пройден.

Таблица 3.4 – Результат получения элементов из отношения со статусом 1

ID	doc_id	status
7	df6286ba-9993-4a15-928b-2ef0652dd699	1

Запрос вида `select * from getTasksReady(2::int2);`.

Ожидание: будут получены элементы отношения со статусом равным 2.

Результат приведен в таблице 3.5.

Таблица 3.5 – Результат получения элементов из отношения со статусом 1

ID	doc_id	status
9	5d92a123-b3a8-4f6d-b449-dc07913f60be	2

Тест был успешно пройден.

Запрос вида `select * from getTasksReady(0::int2);`.

Ожидание: будут получены элементы отношения со статусом равным 0.

Результат приведен в таблице 3.6.

Таблица 3.6 – Результат получения элементов из отношения со статусом 0

ID	doc_id	status
1	710eb9fc-1118-4a66-837e-8e3d2d027a68	0
2	710eb9fc-1118-4a66-837e-8e3d2d027a67	0
3	710eb9fc-1118-4a66-837e-8e3d2d027a65	0
4	710eb9fc-1118-4a66-837e-8e3d2d027a61	0
5	5d92a123-b3a8-4f6d-b449-dc07913f60be	0

Тест был успешно пройден.

3.7.3 Тестирование разработанных триггеров базы данных

Воспользуемся отношением очереди, представленном в таблице 3.2. Рассмотрим следующие варианты использования разработанного триггера `addToQueue`:

1. Попытка загрузки документа с подстрокой «VIP» в названии.
2. Попытка загрузки документа без подстроки «VIP».

3.7.4 Попытка загрузки документа с рассматриваемой подстрокой

Вид запроса представлен в листинге 3.1

Листинг 3.1 – Запрос загрузки документа с подстрокой «VIP»

```
INSERT INTO documents (id,page_count, document_name,
   checks_count, creator_id, creation_time, has_passed)
VALUES ('04570f8a-1f94-4e62-9fbd-02a7beb8ba24',10,
   'VIP_Document', 5, 2, '2024-05-20 12:00:00', true)
```

Ожидание: Отношение очереди останется неизменным.

Результат состояние очереди осталось неизменным.

Тест был успешно пройден.

3.7.5 Попытка загрузки документа без рассматриваемой подстроки

Ожидание: в очереди появится новый документ.

Вид запроса представлен в листинге 3.2 Состояние очереди после запроса представлено в таблице 3.7.

Листинг 3.2 – Запрос загрузки документа без подстоки «VIP»

```
INSERT INTO documents (id,page_count, document_name,
   checks_count, creator_id, creation_time, has_passed)
VALUES ('99c50e01-8c64-4766-9cdb-9ff087486a5d',10,
   'SOME_Document', 5, 2, '2024-05-20 12:00:00', true)
```

Тест был успешно пройден.

Таблица 3.7 – Таблица отношения очереди после добавления документа

ID	**doc_id**	**status**
1	710eb9fc-1118-4a66-837e-8e3d2d027a68	0
2	710eb9fc-1118-4a66-837e-8e3d2d027a67	0
3	710eb9fc-1118-4a66-837e-8e3d2d027a65	0
4	710eb9fc-1118-4a66-837e-8e3d2d027a61	0
5	5d92a123-b3a8-4f6d-b449-dc07913f60be	0
7	df6286ba-9993-4a15-928b-2ef0652dd699	1
9	5d92a123-b3a8-4f6d-b449-dc07913f60be	2
11	99c50e01-8c64-4766-9cdb-9ff087486a5d	0

3.8 Реализация приложения

При загрузке документа сначала сохраняется сам файл документа, после чего в базу данных записываются метаданные файла (время отправки, число проверок, имя файла). После чего данные отправляются на сервер тестирования, где каждая страница отчета преобразуется в файл портативной сетевой графики (англ. Portable network graphic, сокр. png). После чего на каждой странице решается задача детекции с помощью нейросети YOLOv8 [21]. После чего каждый из детектируемых изображений попадает на обработку списку обработчиков, которые, в зависимости от класса изображения(полученного до этого в задаче детекции) осуществляют проверку по соответствующим критериям. В случае указания постфикса VIP при загрузке файла считается что файл проверяется вне очереди и будет проверен системой сразу, иначе он будет добавлен в очередь файлов и помечен как готовый к проверке (поле status), после чего он будет проверен системой при запуска программы с помощью утилиты cron [22].

На рисунке 3.2, представлена диаграмма последовательности действий при работе системы.

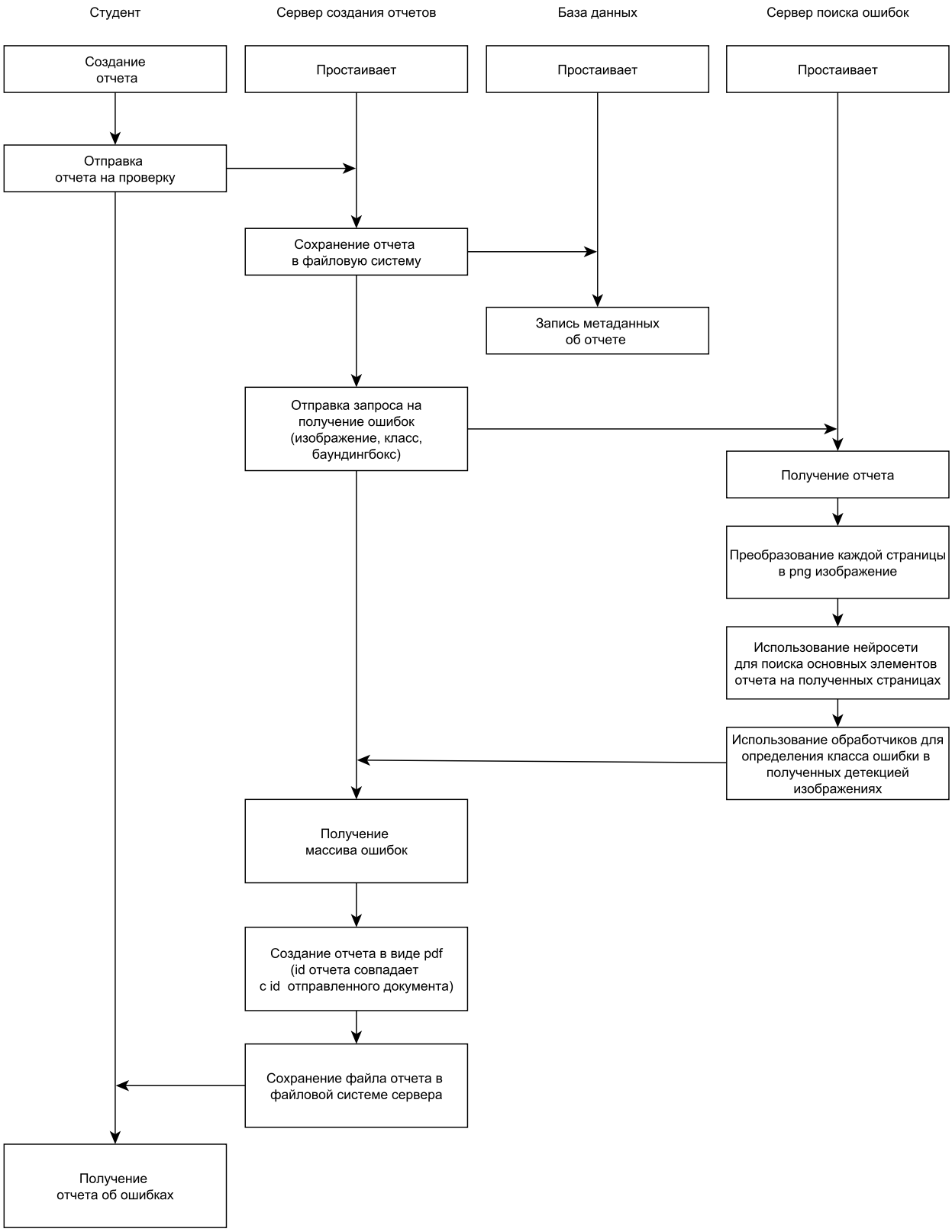

Рисунок 3.2 – Диаграмма последовательности действий получение отчета об ошибках работы студента

3.9 Описание программного интерфейса

Для описания интерфейса взаимодействия с приложением была использована библиотека go-swagger, а также приложение postman [23; 24]. На рисунках 3.3–3.4, представлен интерфейс доступа к сущностям базы данных.

Для взаимодействия с типами фрагментов необходима роль нормоконтроллера или админа, в интерфейсе предоставлены следующие операции:

1. Создание нового типа фрагментов.
2. Получение всех типов фрагментов, созданных текущим вошедшим в систему пользователем.
3. Удаление типа фрагмента по его уникальному идентификатору, данная операцию может совершить только пользователь с ролью «админ».
4. Получение нескольких фрагментов по их уникальным идентификаторам.
5. Получение всех существующих в базе данных типов фрагментов.

Для взаимодействия с фрагментами необходима роль нормоконтроллера или админа, в интерфейсе предоставлены следующие операции:

1. Создание нового фрагмента.
2. Регистрация проверки данного фрагмента для его валидации.
3. Получение всех существующих в базе данных типов фрагментов.
4. Получение всех фрагментов, созданных текущим вошедшим в систему пользователем.
5. Получение фрагмента по его уникальному идентификатору.

Для взаимодействия с документами представлены следующие операции:

1. Получение файла документа по уникальному идентификатору.
2. Получение метаданных обо всех документах, созданных текущем пользователем.

3. Принятие решения о верности данного документа или нет (нормоконтроллер принял документ).

4. Создание и сохранение отчета об ошибках, метаданных и файла документа в системе.

5. Получение файла отчета об ошибках по уникальному идентификатору (уникальные идентификаторы документа и отчета по нему совпадают).

Для регистрации пользователей представлены следующие операции:

1. Зарегистрироваться в системе (по умолчанию выдается роль студента).

2. Войти в систему с использованными при регистрации данными.

По умолчанию при регистрации пользователь получает роль «студент», для ее изменения необходимо, чтобы пользователь с ролью «админ» изменил его роль.

Для взаимодействия с пользователями необходима роль пользователя «админ» ,в интерфейсе представлены следующие операции:

1. Получение всей информации о пользователях зарегистрированных в системе.

2. Изменение роли пользователя по его уникальному логину.

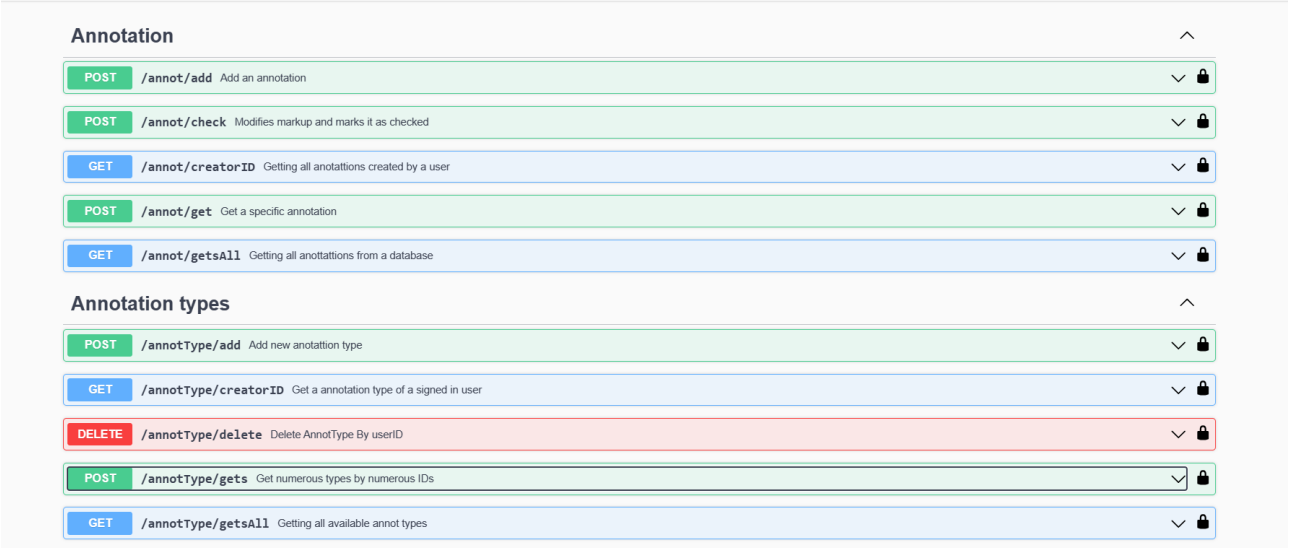

Рисунок 3.3 – Интерфейс взаимодействия с приложением (начало)

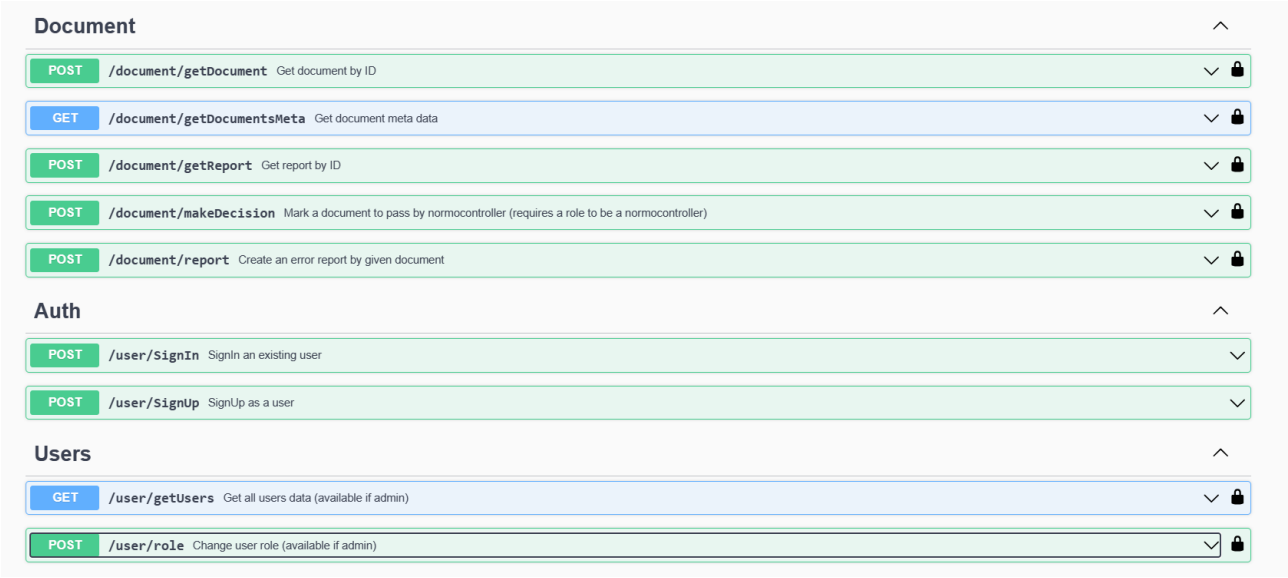

Рисунок 3.4 – Интерфейс взаимодействия с приложением (продолжение)

Вывод

В данном разделе был описан язык программирования, обоснован и совершен выбор СУБД. А также реализована ролевая модель на уровне базы данных, описано тестирование всех разработанных на стороне базы данных функций и триггеров, а также описан интерфейс доступа к базе данных.

4 Исследовательская часть

В данном разделе будет описана зависимость времени обработки запроса на получение разметки от числа запросов в секунду, а также сравнение времени обработки запроса с использованием кеширования и без использования кеширования. Также будет описаны технические характеристики устройства, на котором проводились замеры и приведен анализ полученных результатов.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее.

1. Процессор Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 МГц, ядер: 6, логических процессоров: 12.
2. Оперативная память: 16 ГБайт.
3. Операционная система: Майкрософт Windows 10 Pro [25].
4. Использованная подсистема: WSL2 [26].

При замерах времени ноутбук был включен в сеть электропитания.

4.2 Зависимость скорости обработки запроса от числа запросов в секунду

Для данного исследования в базу данных было сгенерировано и добавлено 20000 размеченных изображений от одного пользователя.

Для создания данного отчета, был реализована программа на языке `Python`, в данной программе на каждый запрос создавался поток, после чего каждую секунду потоки замеряли время в секундах и сохраняли полученные данные его в список, см. листинг Г.1.

Данным образом был отправлен запрос на получение сущности фрагмента, заметим что в данной реализации изображение хранится в виде байт в столбце базы данных.

Результаты измерений представлены в таблице 4.1.

Таблица 4.1 – Зависимость времени обработки запроса от числа запросов в секунду

Число запросов в секунду	Время обработки (с)
10	0.0132
20	0.0164
30	0.0179
40	0.0190
50	0.0216
60	0.0255
70	0.0409
80	0.0263
90	0.0290
100	0.0276
110	0.0302
120	0.0307
130	0.0343
140	0.0409
150	0.0436
160	0.0537
170	0.0571
180	0.0565
190	0.0569
200	0.0571
210	0.0693
220	0.0760
230	0.0730
240	0.0822
250	0.1251
260	0.1151
270	0.1028
280	0.1092
290	0.1103
300	0.1154

По информации из таблицы 4.1 был построен график на рисунке 4.1.

Рисунок 4.1 – График зависимости скорости обработки запроса от числа поступаемых запросов в секунду

4.3 Зависимость скорости обработки запроса от числа запросов в секунду

С использованием JMeter случайно генерировался id разметки для получения, после чего для получения данной разметки генерировались запросы, число клиентов для генерации запросов было равно 100, на каждый клиент выделялся отдельный поток, число запросов от одного клиента равно 1000, запросы одного клиента поступали через каждые 0.1 секунды, все было обработано 99000 запросов [27].

В качестве кеша было использовано хранилище в оператирвной памяти redis [28]. При запросе в случае отсутствия данных по данному id в кеше данные считывались в него, при удалении фрагмента из базы данных данный фрагмент также удалялся из из хранилища redis. В настройках redis

максимальный объем хранимых данных был указан в 1 гигабайт, алгоритм вытеснения с наиболее давним обращением (англ. Least Recently Used сокр. lru).

При использовании кеша сначала происходило обращение к кешу, в случае если необходимого запроса там не оказывалось происходило обращение к базе данных и полученный ответ помещался в кеш.

В таблице 4.2, представлено сравнение двух реализаций, при этом в таблице было всего 15000 записей, в конце обработки запросов в кеше осталось 984 значения.

В результате анализа таблицы 4.2, был получен график 4.2.

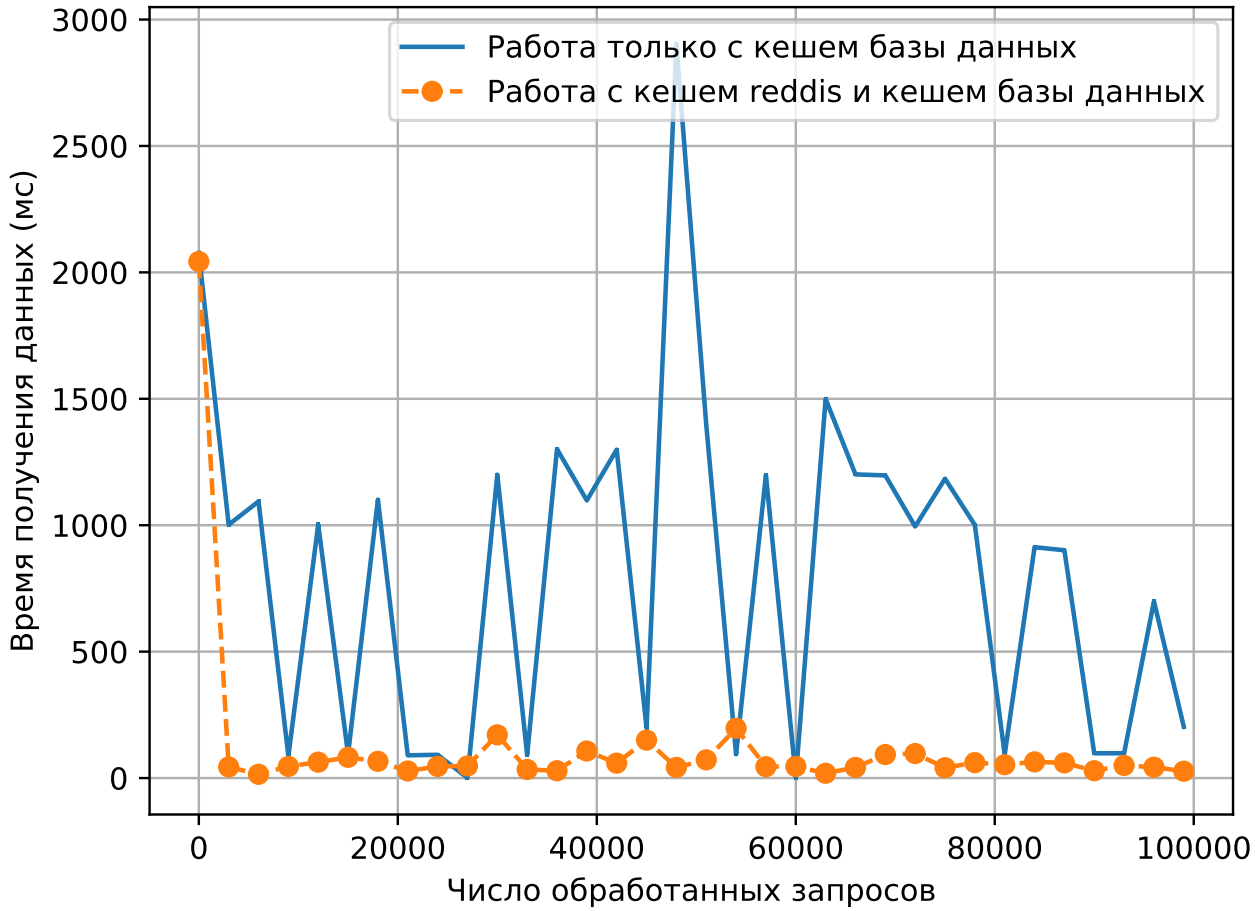

Рисунок 4.2 – График зависимости скорости обработки запроса от наличия кеша и числа обработанных запросов

Вывод

Из графика на рисунке 4.1 можно сделать вывод что среднее время обработки запроса увеличивается с ускорением при увеличении числа запросов в секунду. При увеличении числа запросов с 50 в 3 раза, среднее время

Таблица 4.2 – Зависимость времени обработки запроса от наличия кеша и числа обработанных запросов

Число обработанных запросов	Время обработки запроса без кеша (мс)	Время обработки запроса с кешем (мс)
0	2077	2043
3000	1002	44
6000	1094	15
9000	87	46
12000	1003	63
15000	94	82
18000	1101	66
21000	90	28
24000	92	46
27000	1	48
30000	1200	171
33000	92	34
36000	1301	29
39000	1098	107
42000	1299	59
45000	199	150
48000	2904	42
51000	1398	72
54000	94	197
57000	1198	45
60000	2	46
63000	1499	19
66000	1201	42
69000	1197	93
72000	995	97
75000	1184	41
78000	1001	61
81000	93	53
84000	913	64
87000	901	60
90000	98	29
93000	98	50
96000	699	43
99000	203	27

обработки увеличилось в 2.01 раз, что связано с увеличением нагрузки на базу данных и конкуренции различных пользователей за ресурсы системы.

Из графика на рисунке 4.2 можно сделать вывод, что использование кеша ускорило время обработки запросов. Изначально, до того как запрос попал в кеш время обработки запросов сопоставимо, однако при обнаружении данных запроса в кеше скорость обработки значительно возрастает. При отсутствии объекта в кеше (0 обработанных запросов) время обработки данных с кешем меньше в 1.01 раз. При обнаружении требуемых данных в кеше, при 51000 обработанных запросов время обработки меньше в 19.41 раз. Это связано с большей скорости получения информации из хранилища в оперативной памяти (кеша).

ЗАКЛЮЧЕНИЕ

В результате исследования было определено что время обработки запроса к системе зависит от числа запросов секунду и наличия кеша. Кеш дает значительный выигрыш при сопоставимому числу объектов в нем и во всем множестве. При использовании кеша при 51000 обработанных объектах время обработки запроса было сокращено в 19.41 раз, что связано с более быстрому получнию данных из хранилища в оперативной памяти.

При исследовании зависимости времени обработки запроса от числа запросов в секунду было выявлено, что при увелечении числа запросов в секунду с 50 до 150 среднее время обработки увеличилось в 2.01 раз, что связано с конккуренцией пользователей за ресурсы системы.

Поставленная цель: разработка базы данных обогащения обучающей выборки для автоматизированной проверки отчета на соответствие нормативным требованиям была выполнена.

Для поставленной цели были выполнены все задачи:

1. проанализированы существующие решения;
2. формализована задача и определен необходимый функционал;
3. проанализированы способы хранения данных и системы управления базами данных, выбрана подходящая система для поставленной цели;
4. спроектирована база данных, описаны ее сущности и связи;
5. разработана база данных;
6. исследована зависимость времени выполнения запроса от числа получаемых запросов в секунду;
7. произведено сравнение скорости обработки запросов реализаций с кешем и без.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ВКР ВУЗ [Электронный ресурс]. — URL: http://www.vkr-vuz.ru/ (дата обращения: 27.11.2023).
2. TestVkr [Электронный ресурс]. — URL: https://vkr.bmstu.ru/ (дата обращения: 27.11.2023).
3. PdfTest [Электронный ресурс]. — URL: https://applitools.com/blog/automate-pdf-testing/ (дата обращения: 27.11.2023).
4. *Дж. Д. К.* Введение в системы баз данных: 8-е издание //. — — «Вильямс», 2006. — С. 1328.
5. *К. В. В.* Базы данных, проектирование, программирование, управление и администрирование //. — — Издательство «Лань», 2020.
6. Физические модели баз данных [Электронный ресурс]. — URL: https://intuit.ru/studies/courses/1001/297/lecture/7415?page=5&ysclid=lur73sp2bk237864836 (дата обращения: 08.04.2024).
7. Обзор баз данных временных рядов. — URL: https://cyberleninka.ru/article/n/obzor-baz-dannyh-vremennyh-ryadov (дата обращения: 09.05.2024).
8. *Д. К. С.* Основы современных баз данных //. — — Центр Информационных Технологий, 1998.
9. Анализ популярных реляционных систем управления базами данных (2022 г.) [Электронный ресурс]. — URL: https://drach.pro/blog/hi-tech/item/196-popular-relational-dbms-2022 (дата обращения: 09.05.2024).
10. PostgreSQL [Электронный ресурс]. — URL: https://www.postgresql.org/ (дата обращения: 09.05.2024).
11. Postgres Pro Certified [Электронный ресурс]. — URL: https://postgrespro.ru/products/postgrespro/certified (дата обращения: 09.05.2024).
12. SQL Server 2022 [Электронный ресурс]. — URL: https://www.microsoft.com/ru-ru/sql-server/sql-server-2022 (дата обращения: 09.05.2024).

13. MySQL [Электронный ресурс]. — URL: https://www.mysql.com/ (дата обращения: 09.05.2024).

14. ФСТЭК России приостановила действие 56 сертификатов [Электронный ресурс]. — URL: https://www.anti-malware.ru/news/2022-03-24-111332/38397 (дата обращения: 09.05.2024).

15. Роли уровня сервера [Электронный ресурс]. — URL: https://learn.microsoft.com/ru-ru/sql/relational-databases/security/authentication-access/server-level-roles?view=sql-server-ver16 (дата обращения: 09.05.2024).

16. MySQL Roles – All You Need to Know [Электронный ресурс]. — URL: https://mysqlcode.com/mysql-roles/ (дата обращения: 09.05.2024).

17. gorm [Электронный ресурс]. — URL: https://gorm.io/ (дата обращения: 09.05.2024).

18. Interface in Go [Электронный ресурс]. — URL: https://golangbyexample.com/interface-in-golang/ (дата обращения: 09.05.2024).

19. Discover Packages [Электронный ресурс]. — URL: https://pkg.go.dev/ (дата обращения: 09.05.2024).

20. PyTorch [Электронный ресурс]. — URL: https://pytorch.org/ (дата обращения: 09.05.2024).

21. Ultralytics YOLOv8 Docs [Электронный ресурс]. — URL: https://docs.ultralytics.com/ (дата обращения: 10.11.2023).

22. cron [Электронный ресурс]. — Режим доступа: https://monovm.com/blog/what-is-cron-job/ (дата обращения: 09.05.2024).

23. go-swagger [Электронный ресурс]. — Режим доступа: https://github.com/go-swagger/go-swagger (дата обращения: 09.05.2024).

24. postman [Электронный ресурс]. — Режим доступа: https://www.postman.com/ (дата обращения: 09.05.2024).

25. Windows 10 Pro 2h21 64-bit [Электронный ресурс]. — Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows10 (дата обращения: 28.09.2023).

26. Что такое WSL [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/windows/wsl/about (дата обращения: 28.09.2023).

27. Apache JMeter [Электронный ресурс]. — URL: https://jmeter.apache.org/ (дата обращения: 09.05.2024).

28. redis [Электронный ресурс]. — URL: https://master--redis-doc.netlify.app/docs/about/ (дата обращения: 09.05.2024).

29. ГОСТ 7.32—2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. — М.: Стандартинформ, 2017. — 35 с.

ПРИЛОЖЕНИЕ А

А.1 Основные ошибки в отчетах

В данном разделе будут рассмотрены наиболее часто встречающиеся ошибки, которые совершают студенты при написании различных отчетов.

В целях выявления наиболее часто встречающихся ошибок были опрошены преподаватели, работа которых непосредственно связана с проверкой отчетов студентов.

А.1.1 Общие ошибки

В ГОСТ 7.32 указаны следующие размеры полей: левое — 30 мм, правое — 15 мм, верхнее и нижнее — 20 мм [29]. Выход за границы листа является одной из самых распространенных ошибок.

Каждый объект (таблица, рисунок, схема алгоритма, формула) должен быть подписан и пронумерован, однако более подробно подписи к каждому из них будут рассмотрены в следующих подразделах.

Если таблицу или схему не удается разместить на одной странице, то следует разбить данный объект на несколько частей, каждая из которых должна быть подписана.

А.1.2 Ошибки в тексте

Слова в тексте должны быть согласованы в роде, числе и падеже.

Страницы отчета должны быть пронумерованы, однако, номер на титульном листе не ставится, но он является первой страницей, что означает, что следующая страница должна иметь номер 2.

Ненумерованный заголовок (введение, список литературы, оглавление и т. п.) должен быть выравнен по центру, при этом он состоит только из прописных букв (пример представлен в приложении Б.1), другие варианты оформления являются не соответствующими стандарту.

Абзацный отступ должен быть одинаковым по всему тексту отчета и равен 1,25 см [29]. Любые другие варианты оформления считаются ошибочными.

Возможна потеря научного стиля и переход к публицистике, что является

ошибкой, текст работы должен быть написан на государственном языке в научном стиле.

А.1.3 Ошибки в рисунках

Частой ошибкой является неправильное оформление рисунков. Каждый рисунок должен быть подписан, при этом подпись должна располагаться строго по центру, внизу рисунка. Другое оформление считается ошибочным.

Использование рисунков низкого разрешения является ошибкой. Все рисунки должны быть выполнены в высоком качестве, если обратное не требуется в самой работе.

Некорректный поворот рисунка считается ошибкой. Если рисунок не удается разместить на странице, то допускается повернуть его таким образом, чтобы верх рисунка был ближе к левой части страницы (см. рисунок Б.2).

Ошибки в графиках

Для каждого графика должна существовать легенда, для оформления которой существует два варианта:

– в одном из углов графика находится область, в которой указаны все обозначения;

– в подписи к графику описано каждое обозначение;

другое оформление является ошибкой.

Часто на графиках отсутствуют единицы измерения, что является ошибкой. Должны быть подписаны единицы измерения каждой из осей графика, даже в том случае, если на графике оси подписываются словами, например, если измерение идет в штуках или на оси обозначены времена года (см. рисунок Б.3).

Отчеты могут быть напечатаны в черно-белом варианте, поэтому на графиках должны быть маркеры, которые позволят отличить графики друг от друга даже не в цветом варианте. Отсутствие маркеров считается ошибкой.

При большом количестве графиков на одном рисунке возможна ситуация, при которой невозможно отличить один график от другого, что является ошибкой.

Ошибки в схемах алгоритмов

Если схему не удается разместить на одной странице, то она разбивается на несколько частей, каждая из которых должна быть подписана. Для разделения схемы алгоритма на части используется специальный символ-соединитель, который отображает выход в часть схемы и вход из другой части этой схемы, соответствующие символы-соединители должны содержать одно и то же уникальное обозначение, любые другие варианты оформления являются ошибочными.

Часто вместо символа начала или конца алгоритма используют овал, однако в этом случае должен быть использован прямоугольник с закругленными углами (см. рисунок Б.4).

При использовании символа процесса (прямоугольник) часто используют прямоугольник с закругленными углами (см. рисунок Б.5), что является ошибкой.

При соединении символов схемы алгоритмов не нужны стрелки, если они соединяют символы в направлении слево-направо или сверху-вниз, в остальных случаях символы должны соединяться линиями со стрелкой на конце, отсутствие требуемых стрелок считается ошибкой.

При использовании символа процесса-решение как минимум одна из соединительных линий должна быть подписана (см. рисунок Б.6), однако возможен также вариант, когда подписаны обе линии. Отсутствие пояснений к выходам данного символа является ошибкой.

Часто пояснительный текст пересекается с символами, использующимися для составления схем, что является ошибкой.

А.1.4 Ошибки в таблицах

Каждая таблица должна быть подписана. Наименование следует помещать над таблицей слева, без абзацного отступа в следующем формате: Таблица Номер таблицы - Наименование таблицы. Наименование таблицы приводят с прописной буквы без точки в конце [29]. Другие варианты оформления считаются не соответствующими стандарту.

Таблицу с большим количеством строк допускается переносить на другую страницу. При переносе части таблицы на другую страницу слово «Таб-

лица», ее номер и наименование указывают один раз слева над первой частью таблицы, а над другими частями также слева пишут слова «Продолжение таблицы» и указывают номер таблицы [29]. Любое другое оформление считается ошибочным.

А.1.5 Ошибки в формулах

Каждая формула должна быть пронумерована вне зависимости от того, существует ли ссылка на нее. Нумерация может осуществляться в двух вариантах:

– сквозная нумерация (номер формулы не зависит от раздела, в котором она находится);

– нумерация, зависящая от раздела (в том случае номер формулы начинается с номера раздела);

другое оформление считается ошибкой.

Отсутствие знака препинания после формулы является ошибкой. После каждой формулы должен находиться знак препинания (точка, запятая и т. п.), зависящий от контекста. Если в формуле содержится система уравнений, то после каждого из них (за исключением последнего) ставится запятая, а после последнего — точка, либо запятая (см. рисунок Б.7).

Номер формулы должен быть выравнен по правому краю страницы и находиться по центру формулы (в вертикальной плоскости). Другое оформление нумерации формул считается не соответствующим стандарту.

Если формула вставляется в начале страницы, то часто перед ней может присутствовать отступ, которого быть не должно.

А.1.6 Ошибки в списках

Ненумерованные списки должны начинаться с удлиненного тире (см. рисунок Б.8), другое оформление является ошибочным.

В нумерованных списках после номера пункта обязательно должна стоять скобка (см. рисунок Б.10), использование другого знака считается ошибкой.

В конце каждого пункта списка должен быть знак препинания, от которого зависит первая буква первого слова следующего пункта (см. рисунок Б.9):

– если пункт заканчивается на точку, то первое слово следующего пункта должно начинаться на прописную букву;

– если пункт заканчивается запятой или точкой с запятой, то следующий первое слово следующего слово должно начинаться со строчной буквы;

другое оформление является ошибочным.

A.1.7 Ошибки в списке литературы

Часто при описании одного из источников не указывается одна из составных частей (автор, издательство и т. п.), что является ошибкой.

Также нередко встречаются ссылки на так называемые «препринтовские» издательства (статья еще не вышла), однако была использована в отчете, это считается ошибкой.

ПРИЛОЖЕНИЕ Б

Введение

Рисунок Б.1 – Пример ошибочного оформления ненумерованного заголовка

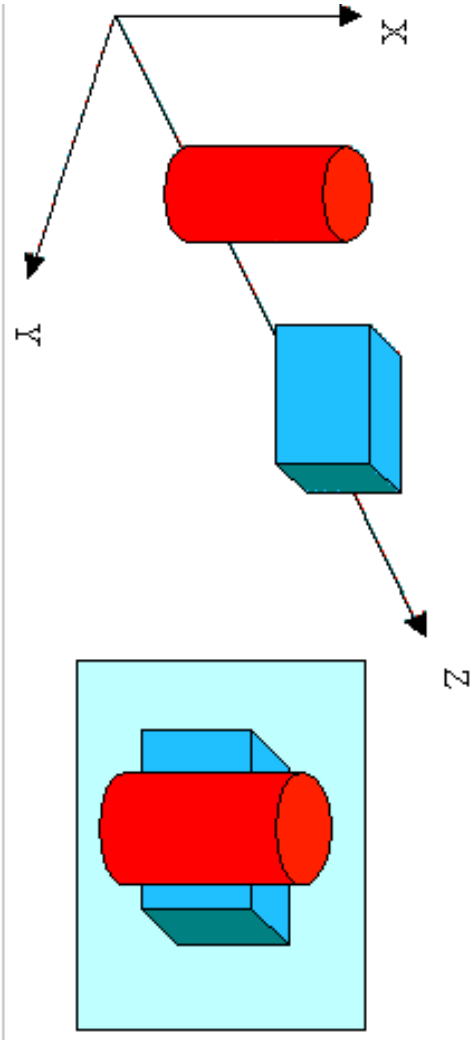

Рисунок 1 - Пример работы Z-буфера

Рисунок Б.2 – Пример ошибочного оформления рисунка — некорректный поворот

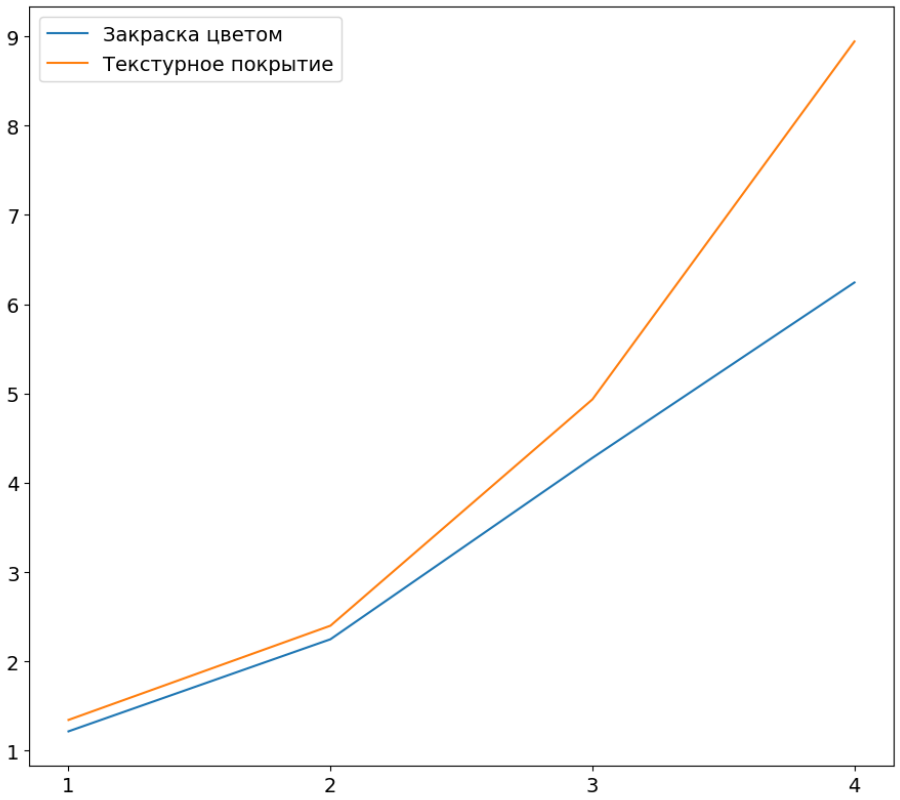

Рисунок Б.3 – Пример ошибочного оформления графика — отсутствуют единицы измерения

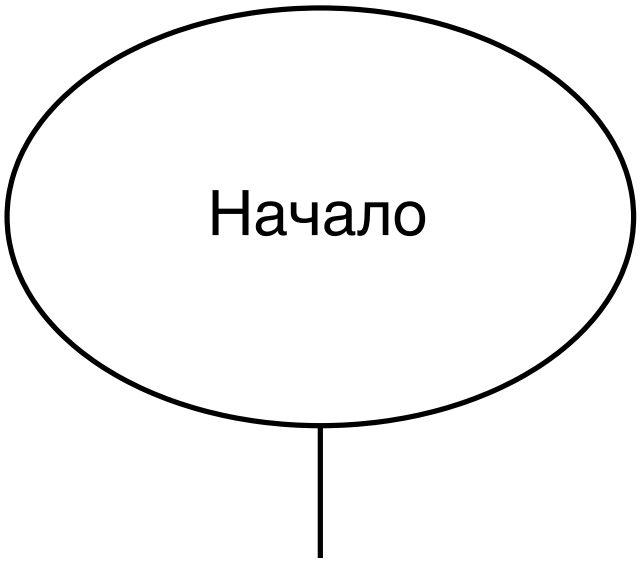

Рисунок Б.4 – Пример ошибочного оформления схемы — некорректный символ начала

Рисунок Б.5 – Пример ошибочного оформления схемы — некорректный символ процесса

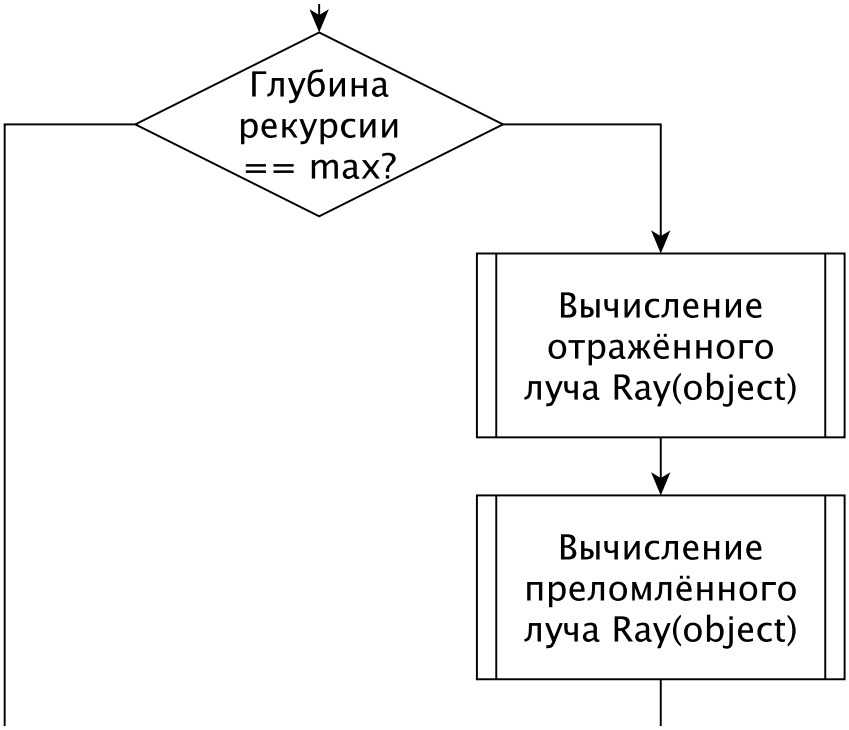

Рисунок Б.6 – Пример ошибочного оформления схемы — не подписана ни одна из веток символа процесса—решение

$$D(i,j) = \begin{cases} 0 & \text{если } i = 0, j = 0 \\ j & \text{если } i = 0, j > 0 \\ i & \text{если } j = 0, i > 0 \\ min(min(D(i,j-1)+1 & \\ \quad D(i-1,j)+1) & \\ \quad D(i-1,j-1) + m(S_1[i], S_2[j]) & \\ \quad \left[\begin{array}{ll} D(i-2,j-2)+1 & \text{если } i > 1, j > 1, \\ & S_1[i-1] == S_2[j-2], \\ & S_1[i-2] == S_2[j-1] \end{array} \right. & \text{иначе} \end{cases} \quad (1)$$

Рисунок Б.7 – Пример ошибочного оформления системы уравнений — отсутствуют знаки препинания после уравнений

- One
- Two
- Three

Рисунок Б.8 – Пример ошибочного оформления ненумерованного списка — некорректный символ перед элементами списка

— Первый,

— Второй,

— Третий.

Рисунок Б.9 – Пример ошибочного оформления ненумерованного списка — некорректный регистр буквы следующего пункта после запятой в предыдущем

1. первый,
2. второй,
3. третий.

Рисунок Б.10 – Пример ошибочного оформления нумерованного списка — некорректный символ после номера элемента списка

ПРИЛОЖЕНИЕ В

Листинг В.1 – Реализация ролевой модели

```
--Создание админа
CREATE ROLE admin WITH
SUPERUSER
NOCREATEDB
CREATEROLE
NOINHERIT
NOREPLICATION
NOBYPASSRLS
CONNECTION LIMIT -1
 LOGIN
 PASSWORD 'admin';

GRANT ALL PRIVILEGES
ON ALL TABLES IN SCHEMA public TO admin;

--Создание добавляющего
CREATE ROLE adder WITH
NOSUPERUSER
NOCREATEDB
NOCREATEROLE
NOINHERIT
NOREPLICATION
NOBYPASSRLS
CONNECTION LIMIT -1
 LOGIN
 PASSWORD 'adder';

GRANT INSERT
ON public.users
TO adder;


--Создание разметчика
CREATE ROLE annoter WITH
NOSUPERUSER
NOCREATEDB
NOCREATEROLE
```

```
38 NOINHERIT
39 NOREPLICATION
40 NOBYPASSRLS
41 CONNECTION LIMIT -1
42  LOGIN
43  PASSWORD 'annoter';
44
45 GRANT INSERT
46 ON public.markups TO annoter;
47
48 GRANT SELECT
49 ON public.documents
50 TO annoter;
51
52
53
54
55 --Создание пользователя (студента)
56 CREATE ROLE student WITH
57 NOSUPERUSER
58 NOCREATEDB
59 NOCREATEROLE
60 NOINHERIT
61 NOREPLICATION
62 NOBYPASSRLS
63 CONNECTION LIMIT -1
64  LOGIN
65  PASSWORD 'student';
66
67
68
69
70 GRANT INSERT
71 ON public.comments
72 TO student;
73
74 GRANT INSERT,SELECT
75 ON public.markups
76 TO student;
77
78
```

```
79
80 --Создание нормоконтроллера
81 CREATE ROLE controller WITH
82 NOSUPERUSER
83 NOCREATEDB
84 NOCREATEROLE
85 INHERIT
86 NOREPLICATION
87 NOBYPASSRLS
88 CONNECTION LIMIT -1
89  LOGIN
90  PASSWORD 'controller';
91
92
93 GRANT  ALL PRIVILEGES
94 ON public.comments
95 TO controller;
96
97 GRANT annoter TO controller;
98
99 GRANT INSERT,SELECT
100 ON public.markup_types
101 TO controller;
102
103 GRANT ALL PRIVILEGES
104 ON public.achievment
105 TO controller;
106
107 --Рабоник очереди
108 CREATE ROLE queue_worker WITH
109 NOSUPERUSER
110 NOCREATEDB
111 NOCREATEROLE
112 NOINHERIT
113 NOREPLICATION
114 NOBYPASSRLS
115 CONNECTION LIMIT -1
116  LOGIN
117  PASSWORD 'queue_worker';
118
119 GRANT SELECT,UPDATE
```

```
ON public.document_queues
TO queue_worker;
```

Листинг В.2 – Реализация создания отношений

```
CREATE TABLE IF NOT EXISTS documents (
    id UUID PRIMARY KEY,
    page_count INT NOT NULL,
    document_name VARCHAR(255) NOT NULL,
    checks_count INT NOT NULL,
    creator_id BIGINT NOT NULL,
    creation_time TIMESTAMP NOT NULL
);

CREATE TABLE IF NOT EXISTS markup_types (
    id serial  PRIMARY KEY,
    description VARCHAR(255) NOT NULL,
    creator_id INT NOT NULL,
    class_name VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS users (
   id serial PRIMARY KEY,
   login VARCHAR(255) UNIQUE NOT NULL,
   password VARCHAR(255) NOT NULL,
   name VARCHAR(255),
   surname VARCHAR(255) NOT NULL,
   role INT NOT NULL,
   user_group VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS markups (
    id serial PRIMARY KEY NOT NULL,
    page_data BYTEA NOT NULL,
    error_bb JSONB DEFAULT '[]' NOT NULL,
    class_label BIGINT NOT NULL,
    creator_id BIGINT NOT NULL
);


CREATE TABLE IF NOT EXISTS document_queues (
    id serial PRIMARY KEY NOT NULL,
    doc_id UUID NOT NULL,
```

```
39      status SMALLINT NOT NULL,
40      CONSTRAINT fk_doc_queue_docs FOREIGN KEY (doc_id) REFERENCES
            documents(id) ON DELETE CASCADE
41  );
42
43
44  CREATE TABLE IF NOT EXISTS comments (
45      id serial PRIMARY KEY NOT NULL,
46      doc_id UUID NOT NULL,
47      description text NOT NULL,
48      creator_id INT NOT NULL,
49      CONSTRAINT fk_comment_docs FOREIGN KEY (doc_id) REFERENCES
            documents(id) ON DELETE CASCADE,
50      CONSTRAINT fk_comment_users FOREIGN KEY (creator_id)
            REFERENCES users(id) ON DELETE CASCADE
51  );
52
53  CREATE TABLE IF NOT EXISTS achievments (
54      id serial PRIMARY KEY NOT NULL,
55      description text NOT NULL,
56      creator_id INT NOT NULL,
57      granted_to_id INT NOT NULL,
58      CONSTRAINT fk_comment_creator FOREIGN KEY (creator_id)
            REFERENCES users(id) ON DELETE CASCADE,
59      CONSTRAINT fk_comment_granted FOREIGN KEY (granted_to_id)
            REFERENCES users(id) ON DELETE CASCADE
60  );
61
62
63  ALTER TABLE markups ADD CONSTRAINT fk_markup_markup_type FOREIGN
        KEY ( class_label ) REFERENCES markup_types( id );
64
65  ALTER TABLE documents ADD CONSTRAINT fk_document_user FOREIGN
        KEY ( creator_id ) REFERENCES users( id );
66
67  ALTER TABLE markups ADD CONSTRAINT fk_markup_user FOREIGN KEY (
        creator_id ) REFERENCES users( id );
68
69  ALTER TABLE markup_types ADD CONSTRAINT fk_markup_types_user
        FOREIGN KEY ( creator_id ) REFERENCES users( id );
70
```

```
71 ALTER TABLE document_queues ADD CONSTRAINT fk_doc_queue_docs
      FOREIGN KEY ( doc_id ) REFERENCES documents( id );
```

Листинг В.3 – Реализация используемых функций

```
1  CREATE  OR REPLACE FUNCTION getTasksReady(task_status int)
2  RETURNS TABLE (
3      id int,
4      doc_id uuid,
5      status SMALLINT
6  )
7  AS $$
8  #variable_conflict use_column
9  BEGIN
10     if (task_status IS NULL) then
11         RETURN QUERY
12         SELECT dq.id, dq.doc_id, dq.status
13         FROM document_queues as dq;
14     else
15         RETURN QUERY
16         SELECT dq.id, dq.doc_id, dq.status
17         FROM document_queues as dq
18         WHERE status = task_status;
19     end if;
20 END;
21 $$ LANGUAGE plpgsql;
```

Листинг В.4 – Реализация используемых триггеров

```
1
2  CREATE OR REPLACE FUNCTION addToQueue()
3  Returns TRIGGER
4  AS $$
5  BEGIN
6      IF NOT EXISTS (
7          SELECT
8              *
9          FROM
10             document_queues
11         WHERE
12             doc_id = NEW.ID AND status = 2
13     ) AND NEW.document_name NOT LIKE '%VIP%'
14     THEN
15         INSERT INTO
```

```
16          document_queues(doc_id,status)
17          VALUES
18          (NEW.ID,0); -- status zero means unchecked
19      END IF;
20      RETURN NEW;
21  END;
22  $$ LANGUAGE plpgsql;
23
24  CREATE TRIGGER insertQueueTrigger AFTER INSERT ON
        public.documents
25  FOR EACH ROW EXECUTE PROCEDURE addToQueue();
```

ПРИЛОЖЕНИЕ Г

Листинг Г.1 – Реализация замера времени отработки запросов

```
import requests
import time
import pandas as pd
import threading
import random

headers = {
    'Authorization': 'Bearer
       eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJRCI6MywiUm9sZSI6MiwiZX
    'Content-Type': 'application/json',
    'Cookie':
       'auth_jwt="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJRCI6MywiUm9
}

urlTest = 'http://localhost:8080/annot/get'

mutex = threading.Lock()

def send_requests(url, n, interval, results,attempt):
    response_times = []
    rand_id = random.randint(1000,14000)
    start_time = time.time()
    response = requests.post(url, headers=headers, json={"id":
       rand_id})
    end_time = time.time()
    if response.status_code == 200:
        response_json = response.json()
        extracted_id = response_json.get('id')
        print("Extracted ID:", extracted_id)
    else:
        print("extract of ID failed", extracted_id)
    mutex.acquire()
    response_time = end_time - start_time
    response_times.append([response_time,n,attempt])
    mutex.release()


```

```
35
36      results.extend(response_times)  # Store the response times
            in the results list
37
38  # Main function to send requests n times with a specified
        interval
39  def send_requests_n_times(url, n_requests, n_times,
        interval_seconds):
40      results = []
41      threads = []
42      for run_id in range(times_block_reqs_ran):
43          for n_time in range(n_times):
44
45              #send_requests(url, n_requests, interval_seconds,
                    results,n_time)
46              thread = threading.Thread(target=send_requests,
                    args=(url, n_times, interval_seconds,
                    results,run_id))
47              threads.append(thread)
48
49
50          for thread in threads:
51              thread.start()
52          for thread in threads:
53              thread.join()
54          time.sleep(1)
55          threads = []
56      return results
57
58
59
60  if __name__ == "__main__":
61      times_block_reqs_ran = 40  # Replace with the total number
            of requests to be sent
62      res_df = pd.DataFrame()
63
64      for  n_requests_per_second in range(10,300 + 1,10):
65          if n_requests_per_second % 10 == 0:
66              print(f"serving {n_requests_per_second} requests per
                    secound")
67
```

```
68          interval_seconds = 1 / n_requests_per_second
69
70          response_times = send_requests_n_times(urlTest,
               times_block_reqs_ran, n_requests_per_second,
               interval_seconds)
71
72          df = pd.DataFrame({'Response Time': response_times})
73          df_exploeded = df.explode('Response Time')
74
75          # Split the response times into 'Value' and 'Time'
               columns
76          all_vals_list = df_exploeded['Response Time'].tolist()
77          Times = [all_vals_list[i] for i in
               range(len(all_vals_list)) if i % 3 == 0]
78          ReqsCount = [all_vals_list[i] for i in
               range(len(all_vals_list)) if i % 3 == 1]
79          Attempt = [all_vals_list[i] for i in
               range(len(all_vals_list)) if i % 3 == 2]
80          df["Times"] = Times
81          df["ReqsCount"] = ReqsCount
82          df["Attempt"] = Attempt
83          df = df.drop('Response Time', axis=1)
84
85          res_df = pd.concat([res_df, df])
86
87  # Save the modified DataFrame to a CSV file
88  res_df.to_csv("response_data.csv", index=False)
```