МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: Сервис локального (на базе нескольких репозиториев) антиплагиата для естественного языка.

Студент гр. 0304 _____ Алексеев Р.В.

Студентка гр. 0304 _____ Говорющенко А.В.

Студент гр. 0304 _____ Максименко Е.М.

Студент гр. 0304 _____ Люлин Д.В.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2023

ЗАДАНИЕ

Студент Алексеев Р.В.

Студентка Говорющенко А.В.

Студент Максименко Е.М.

Студент Люлин Д.В.

Группа 0304

Тема проекта: Разработка сервиса локального антиплагиата для естественного языка.

Исходные данные:

Необходимо реализовать сервис локального антиплагиата на базе нескольких репозиториев для СУБД MongoDB с веб-интерфейсом.

Содержание пояснительной записки:

«Содержание»

«Введение»

«Сценарии использования»

«Модель данных»

«Разработанное приложение»

«Выводы»

«Приложения»

«Литература»

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 26.09.2023

Дата сдачи реферата: 24.12.2023

Дата защиты реферата: 24.12.2023

Студент	_____	Алексеев Р.В.
Студентка	_____	Говорющенко А.В.
Студент	_____	Максименко Е.М.
Студент	_____	Люлин Д.В.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках проекта был разработан сервис локального антиплагиата, работающего с несколькими репозиториями, для естественного языка для СУБД MongoDB. Разработанный сервис имеет веб-интерфейс, позволяющий взаимодействовать с сервисом, выбирать репозитории и смотреть результаты проверок. Исходный код и дополнительную информацию можно найти в репозитории проекта: https://github.com/moevm/nosql2h23-antiplagiat

SUMMARY

As part of the project, a local anti-plagiarism service working with several repositories was developed for natural language for the MongoDB DBMS. The developed service has a web interface that allows you to interact with the service, select repositories and view the results of checks. Source code and additional information can be found in the project repository: https://github.com/moevm/nosql2h23-antiplagiat

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1. Актуальность решаемой проблемы

Существующие сервисы антиплагиата используют для сравнения собственные базы данных с текстами научных работ, но не позволяют проверить организовать проверку на уровень плагиата для нескольких файлов между собой. Проверка относительно друг друга может быть полезна при обработке отчетов студентов по лабораторным и курсовым работам, чтобы выявить случаи списывания у других студентов.

2. Постановка задачи

Необходимо разработать сервис локального антиплагиата, позволяющий проводить анализ файлов из нескольких репозиториев. Сервис должен иметь пользовательский интерфейс для взаимодействия с файлами и репозиториями, а также для просмотра результатов проверки на уровень плагиата.

3. Предлагаемое решение

Было решено разработать веб-приложение, которое позволит взаимодействовать с репозиториями и файлами, настраивать фильтры для анализа файлов на уровень плагиата и просматривать статистику по проверенным файлам.

4. Качественные требования

Требуется разработать приложение с использованием СУБД MongoDB.

1. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1.1. Макет UI

Были разработаны макеты экранов веб-интерфейса сервиса, см. рис. 1 – 6.

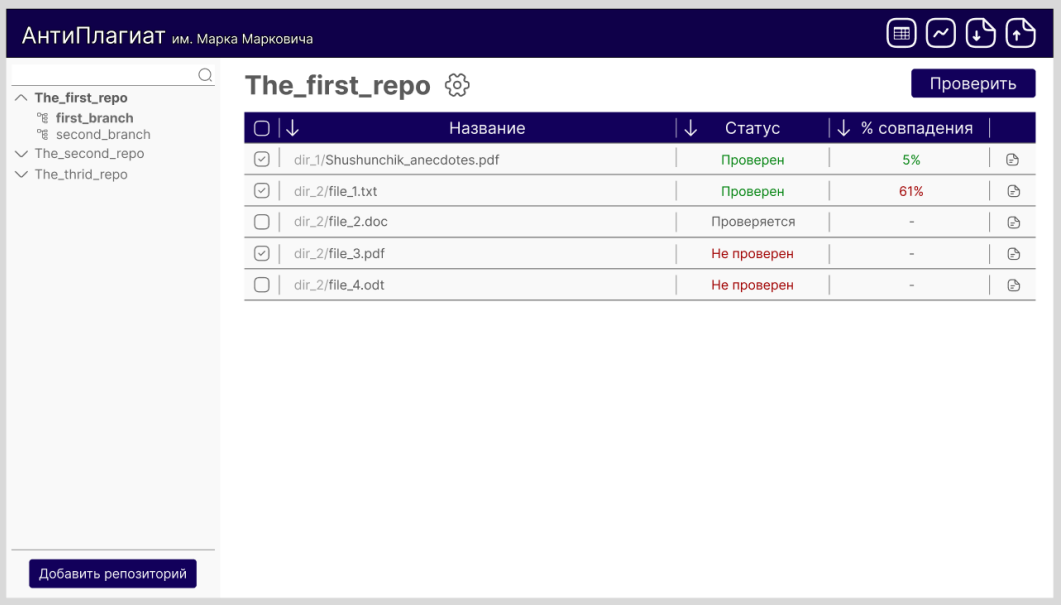

Рисунок 1 – Экран с информацией о файлах репозитория.

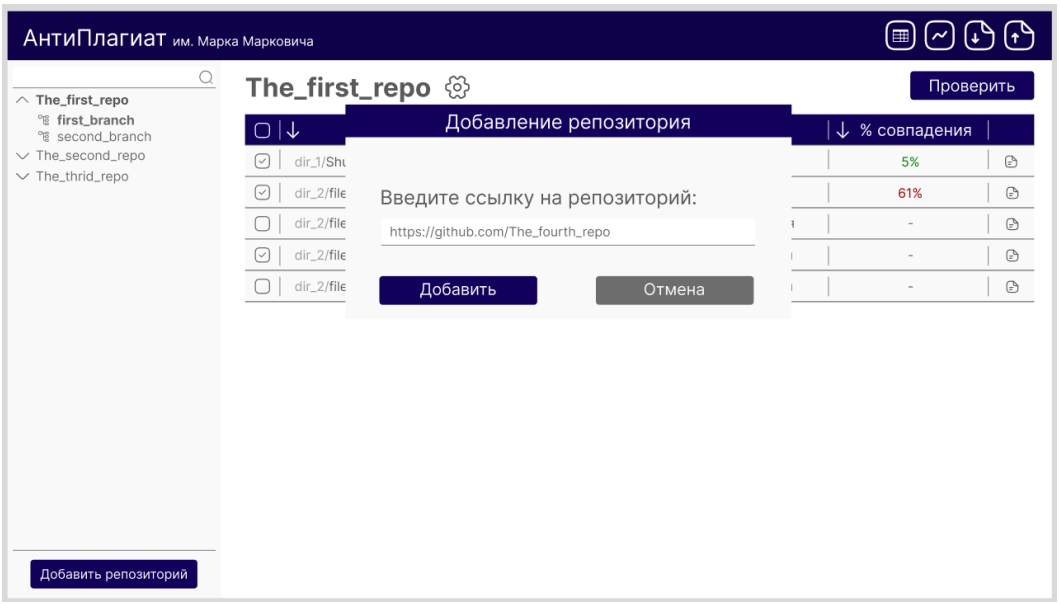

Рисунок 2 – Экран добавления нового репозитория.

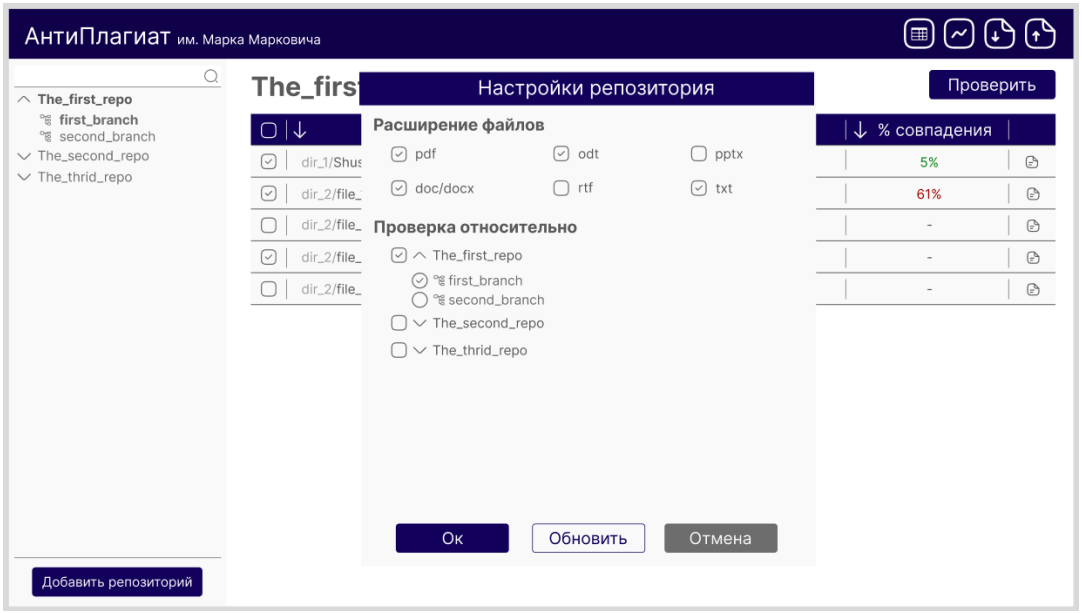

Рисунок 3 – Экран настроек репозитория.

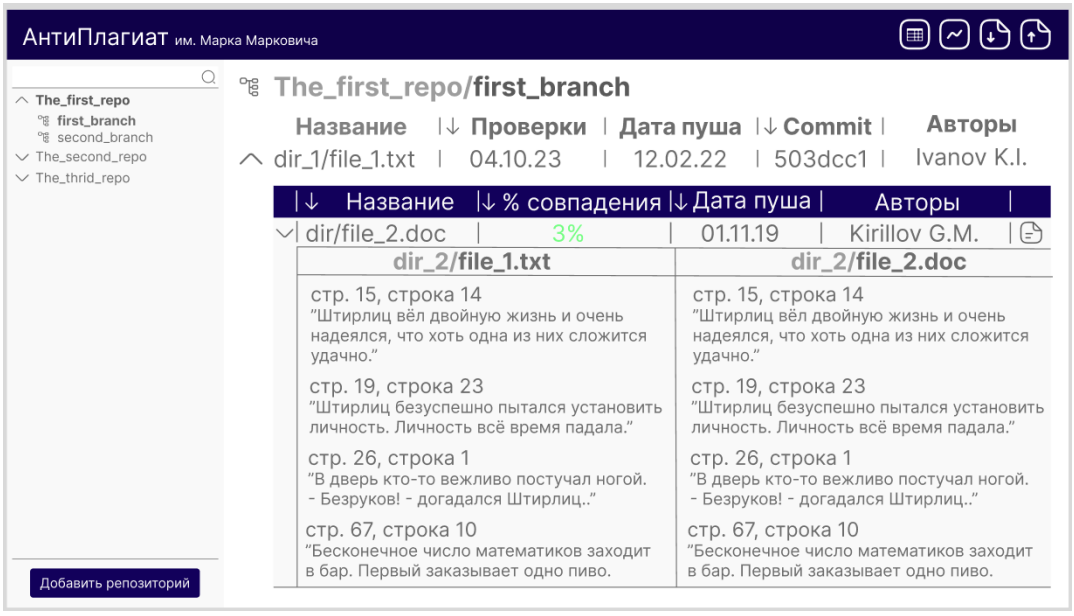

Рисунок 4 – Экран просмотра результатов проверки.

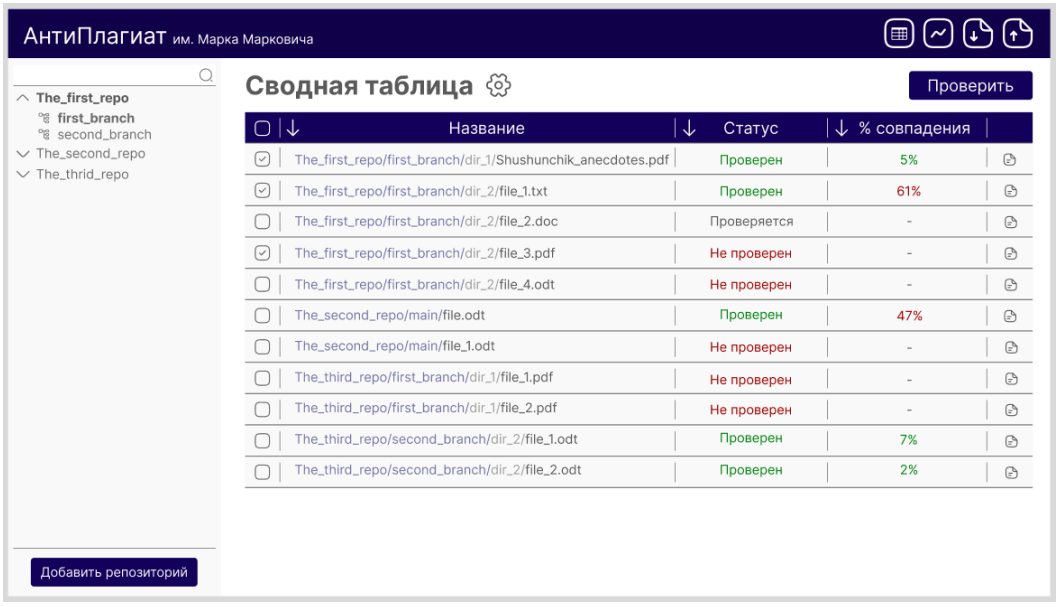

Рисунок 5 – Экран со сводной таблицей.

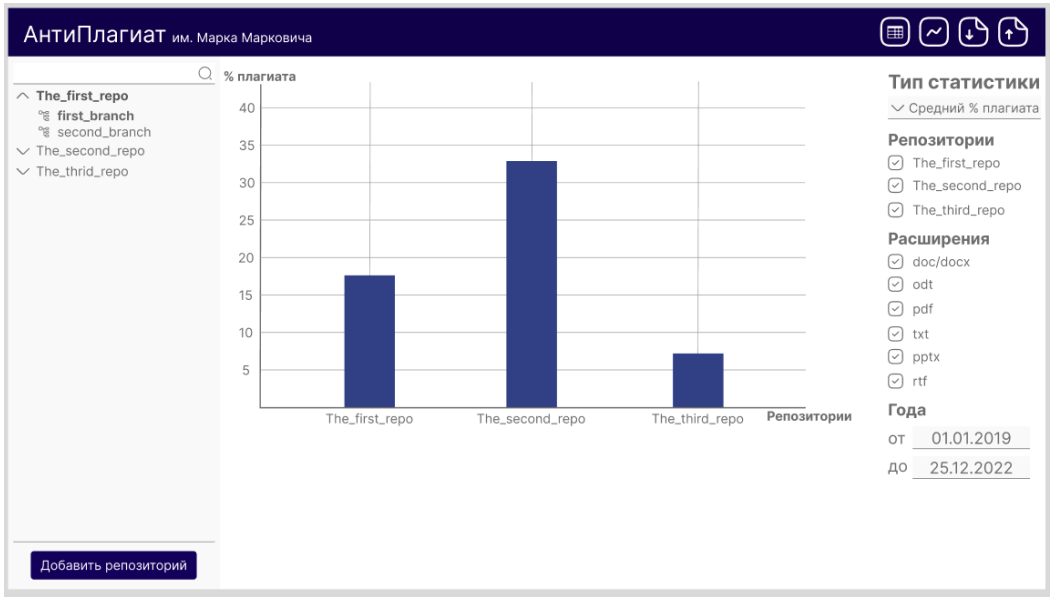

Рисунок 6 – Экран статистики.

1.2. Сценарии использования для задачи

Единственная роль в системе – пользователь, который хочет проверить файлы между собой на уровень плагиата. Пользователь имеет несколько вариантов использования системы:

Массовый импорт:

1. Пользователь нажимает на кнопку импорта в виде файла со стрелкой вниз.
2. Скачивается файл в формате JSON.

Добавление репозитория:

1. Пользователь нажимает на кнопку "Добавить репозиторий".
2. Пользователь вводит ссылку на репозиторий в окне.
3. Пользователь имеет выбор - добавить и отменить.
4. Пользователь нажимает кнопку "Добавить".

Просмотр отчетов проверок:

1. Пользователь нажимает на кнопку в виде документа в строке с названием файла.
2. Пользователь получает выбор между отчетами о проверках выбранного файла.
3. Пользователь нажимает на название отчета.
4. Выбранный отчет скачивается.

Просмотр сводной таблицы:

1. Пользователь нажимает на кнопку в виде таблицы.
2. Открывается страница со сводной таблицей по всем репозиториям.

Просмотр статистики по среднему проценту плагиата в репозиториях:

1. Пользователь нажимает на кнопку в виде графика.
2. Пользователь получает выбор между типами статистики.
3. Пользователь выбирает тип "Средний % плагиата".
4. Открывается гистограмма с средним уровнем плагиата во всех репозиториях.
5. Пользователь выбирает интересующие его репозитории из общего списка.

6. Пользователь выбирает интересующие его расширения файлов.
7. Пользователь выбирает интересующий его временной промежуток, когда был сделан пуш.
8. Гистограмма изменяется с учетом выбранных репозиториев.

Изменение настроек репозитория:

1. Пользователь нажимает на кнопку в виде шестеренки возле названия репозитория.
2. Пользователь выбирает расширения файлов, которые необходимо проверить.
3. Пользователь выбирает репозитории, относительно файлов которых будет производится проверка.
4. Пользователь выбирает ветки в выбранных репозиториях.
5. Пользователь имеет выбор - сохранить настройки, обновить информацию о репозиториях, отменить изменение настроек.
6. Пользователь нажимает кнопку "Ок".

Запуск проверок:

1. Пользователь выбирает файлы при помощи checkbox.
2. Пользователь нажимает на кнопку "Проверить".
3. Система проверяет выбранные файлы.
4. Система сообщает об окончании проверки.
5. Пользователь просматривает отчеты.

Массовый экспорт:

1. Пользователь нажимает на кнопку импорта в виде файла со стрелкой вверх.
2. Пользователь выбирает файл в формате JSON.
3. Пользователь загружает файл в систему.

1.3. Основные операции

Функции добавления и настроек репозитория, просмотра результатов проверок и статистики реализованы при помощи веб-интерфейса.

В предложенной операции чтения и записи используются в равной степени, т.к. необходимо искать совпадения между файлами и сохранять файлы и результаты проверок.

2. МОДЕЛЬ ДАННЫХ

2.1. Нереляционная модель данных

Для хранения репозиториев, коммитов, файлов и результатов проверок созданы отдельные коллекции: *repo, commit, file* и *check* соответственно.

Графическое представление.

Коллекция *repo*:

```
{
    "$name": "repo",
    "name": "string",
    "link": "string",
    "branches": [
        {
            "$name": "branch",
            "name": "string",
            "commits": string[]
        }
    ]
}
```

Коллекция *commit*:

```
{
    "$name": "commit",
    "_id": "string",
    "author": "string",
    "date": "timestamp",
    "files": file::_id[]
}
```

Коллекция *file*:

```
{
    "$name": "file",
    "name": "string",
    "text": "string",
    "commit": "string",
    "data": [
        {
            "$name": "preprocessed_data",
            "text_index": "int",
            "hash": "string"
        }
    ],
    "checks": [
        {
            "$name": "check_result",
            "_id": "check::_id",
            "result": "float"
        }
    ]
```

```
}
```

Коллекция *check*:

```
{
    "$name": "check",
    "date": "timestamp",
    "pairs": [
        {
            "$name": "pair_check",
            "file1": "file::_id",
            "file2": "file::_id",
            "matches": [
                {
                    "matchIndex1": "int",
                    "matchIndex2": "int",
                    "matchLength": "int"
                }
            ],
            "result": "float"
        }
    ]
}
```

Описание назначений коллекций, типов данных и сущностей.

Тип данных «Репозиторий» (коллекция *repo*):

- name - имя репозитория
- branches - ветки
    - name - название ветки
    - commits - коммиты
        - hash - хэш коммита

Тип данных «Коммит» (коллекция *commit*):

- _id - хэш коммита
- author - автор коммита
- date - дата коммита
- files - файлы, измененные в коммите

Тип данных «Файл» (коллекция *file*):

- name - имя файла
- text - содержимое файла

- commit - коммит, в котором был изменен файл в последний раз
- data - триграммы содержимого файла
    - text_index - позиция триграммы в тексте
    - hash - хэш триграммы
- checks - проверки файла
    - result - результат проверки (% плагиата)

Тип данных «Проверка» (коллекция *check*):

- date - дата проверки
- pairs - пара проверяемых файлов
    - file1 - ID первого файла
    - file2 - ID второго файла
    - matches - совпадения:
        - matchIndex1 - индекс совпадения в первом файле
        - matchIndex2 - индекс совпадения во втором файле
        - matchLength - длина совпавшего фрагмента
    - result - результат проверки (% совпадения)

Оценка удельного объема информации, хранимой в модели.

Тип данных "Репозиторий":

- _id - тип ObjectId - 12b
- name - тип String - 32b
- branches - тип Array(Object) - 2 * (320b + 32b) = 704b:
    - name - тип String - 32b
    - commits - тип Array(String) - 10 * 32b = 320b:
        - commit::_id- тип String - 32b

Тип данных "Коммит":

- _id - тип String - 32b
- author - тип String - 32b

- date - тип Timestamp - 8b
- files - Array(Object) - 1 * 12b = 12b:
    - File::_id - тип ObjectId - 12b

Тип данных "Файл":

- _id - тип ObjectId - 12b
- name - тип String - 32b
- text - тип String - 32b
- commit - тип String - 32b
- data - Array(Object) - 200 * (4b + 32b) = 7200b:
    - text_index - тип Int - 4b
    - hash - тип String - 32b
- checks - Array(Object) - 2 * (12b + 8b) = 40b:
    - Check::_id - тип ObjectId - 12b
    - result - тип Double - 8b

Тип данных "Проверка":

- _id - тип ObjectId - 12b
- date - тип Timestamp - 8b
- pairs - Array(Object) - 44b:
    - file1 - тип File::_id - 12b
    - file2 - тип File::_id - 12b
    - matches - тип Array(Object) - 12b:
        - matchIndex1 - тип Int - 4b
        - matchIndex2 - тип Int - 4b
        - matchLength - тип Int - 4b
    - result - тип Double - 8b

Для оценки примем количество репозиториев у пользователя равное 5, каждый репозиторий содержит 2 ветки. В каждой ветке 10 коммитов, каждый

из которых создает N файлов. В каждой ветке 10 файлов, каждый файл проверяется 2 раза и в каждой проверке находится 1 совпадение.

В предложенном решении текст хранится в предварительно обработанном виде, для анализа вычисляются все триграммы (подряд идущие тройки слов), и в БД сохраняются хэши триграмм с индексами в тексте. Примем количество триграмм в файле равное 200. На основе этих данных были рассчитаны фактический и чистый объемы данных, а также избыточность.

Фактический объем данных:

$5 * (repo.\_id + repo.name + 2 * repo.branches) + 5 * 2 * N * (file.\_id + file.name + file.text + file.commit + 200 * (file.data.text\_index + file.data.hash) + file.checks)) + 5 * 2 * N * (commit.\_id + commit.author + commit.date + commit.files) + 5 * 2 * N * (check.\_id + check.date) + 5 * 2 * N * (5 * 2 * N – 1) * check.pairs = 4400 * N^2 + 70440 * N + 3740 = 4400 * N^2 +O(N)$

Чистый объем данных:

Чистый объем вычислен на основе фактического, но из которого исключена избыточность.

$5 * (repo.name + 2 * repo.branches) + 5 * 2 * N * (file.name + file.text + 200 * (file.data.text\_index + file.data.hash) + file.checks)) + 5 * 2 * N * (commit.\_id + commit.author + commit.date) + 5 * 2 * N * (check.date) + 5 * 2 * N * (5 * 2 * N – 1) * check.pairs = 2000 * N^2 + 53600 * N + 800 = 2000 * N^2 +O(N)$

Избыточность предложенной модели:

$(4400 * N^2) / (2000 * N^2) = 2.2$

Для данной модели были определены направления роста при увеличении количества объектов каждой сущности. При увеличении количества репозиториев, коммитов и проверок объем возрастает линейно, а при увеличении количества файлов квадратично.

Запросы к модели, с помощью которых реализуются сценарии использования.

Добавление репозитория:

```
db.file.insertMany( [
    {
        "name": "hello_world/main.js",
        "text": "some text",
        "commit": "410c5b34a218e8a8793525b9b7772362e1ec7561",
        "data": [],
        "checks": []
    },
    ...
] ) -> [ "fileID1", "fileID2", "fileID3", ... ]
db.commit.insertMany( [
    {
        "_id": "410c5b34a218e8a8793525b9b7772362e1ec7561",
        "author": "HypeR (hyperru1lvl@gmail.com)",
        "date": 1699286753,
        "files": [ "fileID1", "fileID3", ... ]
    },
    ...
])
db.repo.insertOne( {
    "link": "https://github.com/moevm/nosql2h23-antiplagiat",
    "name": "moevm/nosql2h23-antiplagiat",
    "branches": [
        {
            "name": "main",
            "commits": [
                "410c5b34a218e8a8793525b9b7772362e1ec7561",
                "503dcc1e48e87239b67bc564e3b4a385026c4212"
            ]
        }
    ]
} )
```

Просмотр репозиториев и веток:

```
db.repo.find()
```

Просмотр файлов в таблице:

```
db.repo.aggregate( [
    { "$unwind": "$branches" },
    { "$unwind": "$branches.commits" },
    {
        "$lookup": {
            "from": "commit",
            "localField": "branches.commits",
            "foreignField": "_id",
            "as": "commitInfo"
        }
    },
    { "$unwind": "$commitInfo" },
    {
```

```
        "$lookup": {
            "from": "file",
            "localField": "commitInfo._id",
            "foreignField": "commit",
            "as": "file"
        }
    },
    { "$unwind": "$file" },
    { "$unwind": "$file.checks" },
    {
        "$lookup": {
            "from": "check",
            "localField": "file.checks",
            "foreignField": "_id",
            "as": "check"
        }
    },
    {
        "$project": {
            "_id": 0,
            "link": 1,
            "name": 1,
            "branch": "$branches.name",
            "commit.hash": "$commitInfo._id",
            "commit.author": "$commitInfo.author",
            "commit.date": "$commitInfo.date",
            "file": 1
        }
    }
] )
```

Запись результатов проверки:

```
db.check.insertOne( {
    "date": 1699286989,
    "pairs": [
        {
            "file1": "fileID1",
            "file2": "fileID3",
            "matches": [
                {
                    "matchIndex1": 10,
                    "matchIndex2": 41,
                    "matchLength": 11
                },
                "result": 32.74
            ]
        },
        ...
    ]
} ) -> checkId
db.file.updateOne( { "_id": "fileID1" }, { "$push": { "checks": {
    "_id": "checkId",
    "result": 64.12
} } } )
db.file.updateOne( { "_id": "fileID3" }, { "$push": { "checks": {
    "_id": "checkId",
    "result": 14.19
```

```
} } } )
...
```

Просмотр результатов проверки файла:

```
db.check.aggregate( [
    { "$unwind": "$pairs" },
    {
        "$lookup": {
            "from": "file",
            "localField": "pairs.file1",
            "foreignField": "_id",
            "as": "file1"
        }
    },
    {
        "$lookup": {
            "from": "file",
            "localField": "pairs.file2",
            "foreignField": "_id",
            "as": "file2"
        }
    },
    {
        "$match": { "$or": [ { "file1._id": "1" }, { "file2._id":
"1" } ] }
    },
    {
        "$project": {
            "date": 1,
            "file1": 1,
            "file2": 1,
            "pairs.result": 1,
            "pairs.matches": 1
        }
    }
] )
```

Просмотр статистики:

```
db.repo.aggregate( [
    { "$unwind": "$branches" },
    { "$unwind": "$branches.commits" },
    {
        "$lookup": {
            "from": "commit",
            "localField": "branches.commits",
            "foreignField": "_id",
            "as": "commitInfo"
        }
    },
    { "$unwind": "$commitInfo" },
    {
        "$lookup": {
            "from": "file",
            "localField": "commitInfo._id",
            "foreignField": "commit",
            "as": "file"
```

```
        }
    },
    { "$unwind": "$file" },
    { "$unwind": "$file.checks" },
    {
        "$lookup": {
            "from": "check",
            "localField": "file.checks",
            "foreignField": "_id",
            "as": "check"
        }
    },
    {
        "$project": {
            "_id": 0,
            "link": 1,
            "name": 1,
            "branch": "$branches.name",
            "commit.hash": "$commitInfo._id",
            "commit.author": "$commitInfo.author",
            "commit.date": "$commitInfo.date",
            "file": 1
        }
    },
    {
        "$group": {
            "_id": "$name",
            "avg": { "$avg": "$file.checks.result" }
        }
    }
] )
```

2.2. Аналог модели данных для SQL СУБД

Для модели данных для MongoDB был создан аналог для SQL СУБД.

Графическое представление.

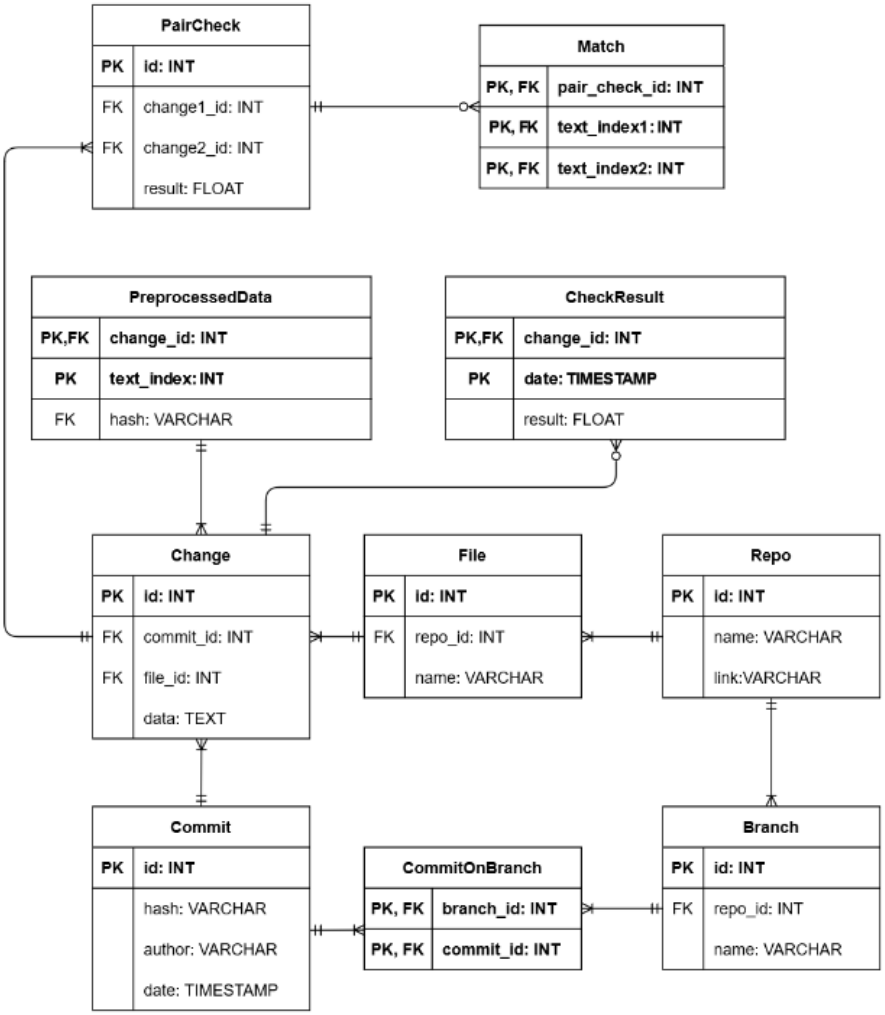

Рисунок 7 – Графическое представление модели для SQL СУБД.

Описание назначений коллекций, типов данных и сущностей.

PairCheck - пары проверенных файлов

- id - ID пары
- change1_id - ID нужной версии первого файла
- change2_id - ID нужной версии второго файла
- result - результат проверки (% совпадения)

Match - совпадения в файлах

- pair_check_id - ID пары файлов
- text_index1 - индекс совпавшего фрагмента из первого файла
- text_index2 - индекс совпавшего фрагмента из второго файла

PreprocessedData - информация о предобработанных данных

- change_id - ID нужной версии файла
- text_index - позиция триграммы в тексте
- hash - хэш триграммы

CheckResult- результаты проверки

- change_id - ID нужной версии файла
- date - дата проверки
- result - результат проверки (% плагиата)

Change - изменения файлов

- id - ID изменения
- commit_id - ID коммита, в котором было внесено изменение
- file_id - ID измененного файла
- data - дата изменения

File - файлы

- id - ID файла
- repo_id - ID репозитория
- name - имя файла

Repo - репозитории

- id - ID репозитория
- name - имя репозитория
- link - ссылка на репозиторий

Commit - коммиты

- id - ID коммита
- hash - хэш коммита
- author - автор коммита
- date - дата коммита

CommitOnBranch - связь коммитов и веток

- branch_id - ID ветки
- commit_id - ID коммита

Branch - ветки

- id - ID ветки
- repo_id - ID репозитория
- name - имя ветки

Оценка удельного объема информации, хранимой в модели.

Примем количество репозиториев, коммитов, веток, файлов и триграмм таким же, как и для нереляционной модели, где N – количество файлов. Тогда найдем фактический и чистый объемы, а также избыточность.

Фактический объем:

PairCheck - 4 * 4 = 16b

Match - 3 * 4 = 12b

PreprocessedData - 2 * 4 + 32 = 40b

CheckResult - 3 * 4 = 12b

Change - 3 * 4 + 32 = 44b

File - 2 * 4 + 32 = 40b

Repo - 4 + 2 * 32 = 68b

Commit - 2 * 4 + 2 * 32 = 72b

CommitOnBranch - 2 * 4 = 8b

Branch - 2 * 4 + 32 = 40b

5 * (*Repo* + 2 * (*Branch* + 10 * *Commit*)) + (2 * 5 * N * (*File* + 2 * *CheckResult*)) + (5 * 2 * N * *CommitOnBranch*) + (5 * 2 * N * *Change*) + (5 * 2 * N * 200 * *PreprocessedData*) + (2 * 5 * N)(2 * 5 * N – 1) * (*PairCheck* +*Match*) = 2400 * N^2 + 80920 * N + 7940 = 2400 * N^2 + O(N)

Чистый объем:

PairCheck - 4b

Match - 2 * 4 = 8b

PreprocessedData - 4 + 32 = 36b

CheckResult - 2 * 4 = 8b

Change - 32b

File - 32b

Repo - 2 * 32 = 64b

Commit - 4 + 2 * 32 = 68b

Branch - 32b

5 * (*Repo* + 2 * (*Branch* + 10 * *Commit*)) + (2 * 5 * N * (*File* + 2 * *CheckResult*)) + (5 * 2 * N * *Change*) + (5 * 2 * N * 200 * *PreprocessedData*) + (2 * 5 * N)(2 * 5 * N – 1) * *PairCheck* = $1200 * N^2 + 7880 * N + 7440 = 1200 * N^2 + O(N)$

Избыточность модели:

$(2400 * N^2) / (1200 * N^2) = 2$

Для данной модели были определены направления роста при увеличении количества объектов каждой сущности. У *PairCheck* и *Match* наблюдается квадратичный рост с увеличением файлов, т.к. эти сущности связаны с парами файлов. Всего пар N * (N - 1) / 2. У остальных сущностей рост линейный.

Запросы к модели, с помощью которых реализуются сценарии использования.

Добавление репозитория:

```
1) INSERT INTO Repo(name, link) VALUES ("repo1", "link1");
2) INSERT INTO Commit(hash, author, date) VALUES ("aead123fa", "simoesh", 1698083590), ... , (...);
3) Аналогичная вставка в таблицу File;
4) Аналогичная вставка в таблицу Change;
5) Аналогичная вставка в таблицу Branch;
6) Аналогичная вставка в таблицу CommitOnBranch;
7) Аналогичная вставка в таблицу PreprocessedData;
```

Просмотр репозиториев и веток:

```
SELECT Repo.name, Branch.name FROM Repo JOIN Branch ON Branch.repo_id = Repo.id;
```

Просмотр файлов в таблице:

```
SELECT File.name, CheckResult.result FROM CheckResult
       JOIN Change ON CheckResult.change_id = Change.id
       JOIN File ON Change.file_id = File.id
       JOIN Commit ON Change.commit_id = Commit.id
```

```
            JOIN CommitOnBranch ON CommitOnBranch.commit_id = Commit.id
            JOIN CommitOnBranch.branch_id = Branch.id
            JOIN Repo ON Repo.id = Branch.Repo
            JOIN (
                SELECT File.name, MAX(CheckResult.date) AS latest_date
                    FROM CheckResult
                    JOIN Change ON CheckResult.change_id = Change.id
                    JOIN File ON Change.file_id = File.id
                    JOIN Commit ON Change.commit_id = Commit.id
                    JOIN CommitOnBranch ON CommitOnBranch.commit_id =
Commit.id
                    JOIN CommitOnBranch.branch_id = Branch.id
                    JOIN Repo ON Repo.id = Branch.Repo
                GROUP BY File.name
                WHERE Repo.name = "repo1" AND Branch.name = "branch1" )
tmp
            ON tmp.latest_date = CheckResult.date AND File.name =
tmp.name
        GROUP BY File.name WHERE Repo.name = "repo1" AND Branch.name =
"branch1";
```

Запись результатов проверки:

```
    1) SELECT PreprocessedData.change_id, PreprocessedData.text_index,
PreprocessedData.hash
           FROM PreprocessedData
               JOIN Change ON CheckResult.change_id = Change.id
               JOIN File ON Change.file_id = File.id
               JOIN Commit ON Change.commit_id = Commit.id
               JOIN CommitOnBranch ON CommitOnBranch.commit_id =
Commit.id
               JOIN CommitOnBranch.branch_id = Branch.id
               JOIN Repo ON Repo.id = Branch.Repo
           WHERE Repo.name = "repo1" AND Branch.name = "branch1" AND
File.name LIKE '.*.odt';
    2) INSERT INTO PairCheck(change1_id, change2_id, result) VALUES
(12454, 39537, 34.57), ..., (12454, 9977, 12.59);
    3) INSERT INTO Match(pair_check_id, text_index1, text_index2)
VALUES (51256, 789, 8745), ... , (...);
    4) INSERT INTO CheckResult(change_id, date, result) VALUES (12454,
1698083590, 45.65);
```

Просмотр результатов проверки файла:

```
    SELECT File.name, Repo.name, Branch.name,
               CheckResult.result, Commit.author,
               Commit.date, PairCheck.result, Change.data,
               Match.text_index1, Match.text_index2
            FROM CheckResult
            JOIN Change ON CheckResult.change_id = Change.id
            JOIN File ON Change.file_id = File.id
            JOIN Commit ON Change.commit_id = Commit.id
            JOIN CommitOnBranch ON CommitOnBranch.commit_id = Commit.id
            JOIN CommitOnBranch.branch_id = Branch.id
            JOIN Repo ON Repo.id = Branch.Repo
            JOIN PairCheck ON Change.id = PairCheck.change1_id
            JOIN Match ON Match.pair_check_id = PairCheck.id
        WHERE File.name = "file1";
```

Просмотр статистики:

```
SELECT AVG(CheckResult.result), Repo.name
       FROM CheckResult
       JOIN Change ON CheckResult.change_id = Change.id
       JOIN File ON Change.file_id = File.id
       JOIN Commit ON Change.commit_id = Commit.id
       JOIN CommitOnBranch ON CommitOnBranch.commit_id = Commit.id
       JOIN CommitOnBranch.branch_id = Branch.id
       JOIN Repo ON Repo.id = Branch.Repo
       JOIN PairCheck ON Change.id = PairCheck.change1_id
       JOIN Match ON Match.pair_check_id = PairCheck.id
    GROUP BY Repo.name WHERE File.name LIKE '.*.pdf' AND
CheckResult.Date < 1698083590 AND CheckResult.Date > 1698094560;
```

2.3. Сравнение моделей

Было произведено сравнение предложенной модели для MongoDB и её аналога для SQL СУБД.

Из-за затрат на дублирование данных в нереляционной модели избыточность данной модели выше, чем у реляционной – 2.2 у нереляционной и 2 у реляционной. Таким образом данные в модели для MongoDB занимают больший объем.

Реляционная модель требует большего числа запросов и более сложных запросов по структуре, чем нереляционная при аналогичной цели запроса, т.к. необходимо соединять несколько таблиц.

На основании сравнения сделан вывод, что обе модели одинаково подходят для решения поставленной задачи, т.к. реляционная модель обладает меньшей избыточностью, а для нереляционной модели необходимы более простые запросы.

3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

3.1. Описание

Back-end представляет из себя JavaScript-приложение.

Front-end – web-приложение, которое использует API back-end для доступа к данным.

3.2. Использованные технологии

В качестве СУБД использована MongoDB.

Для back-end использован JavaScript.

Для front-end использованы TypeScript, Vue.

3.3. Снимки экрана приложения

Снимки экрана рабочего приложения приведены на рис. 8–14.

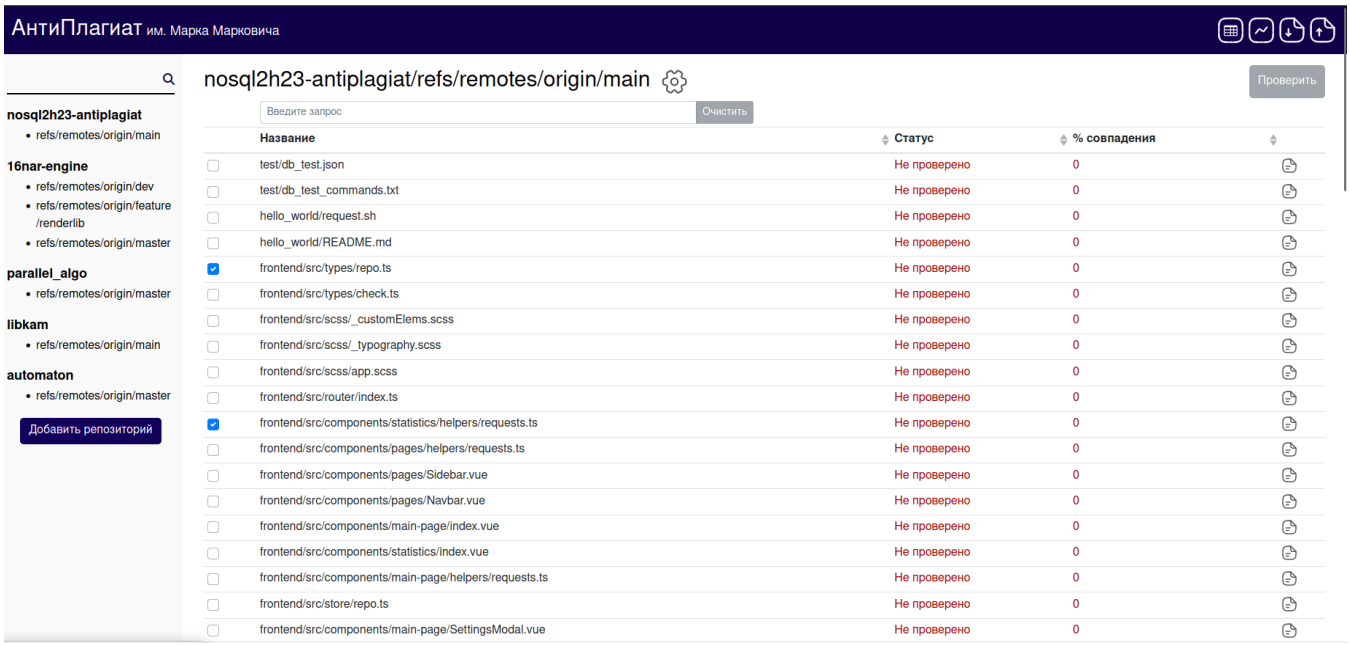

Рисунок 9. Экран с файлами с ветки репозитория.

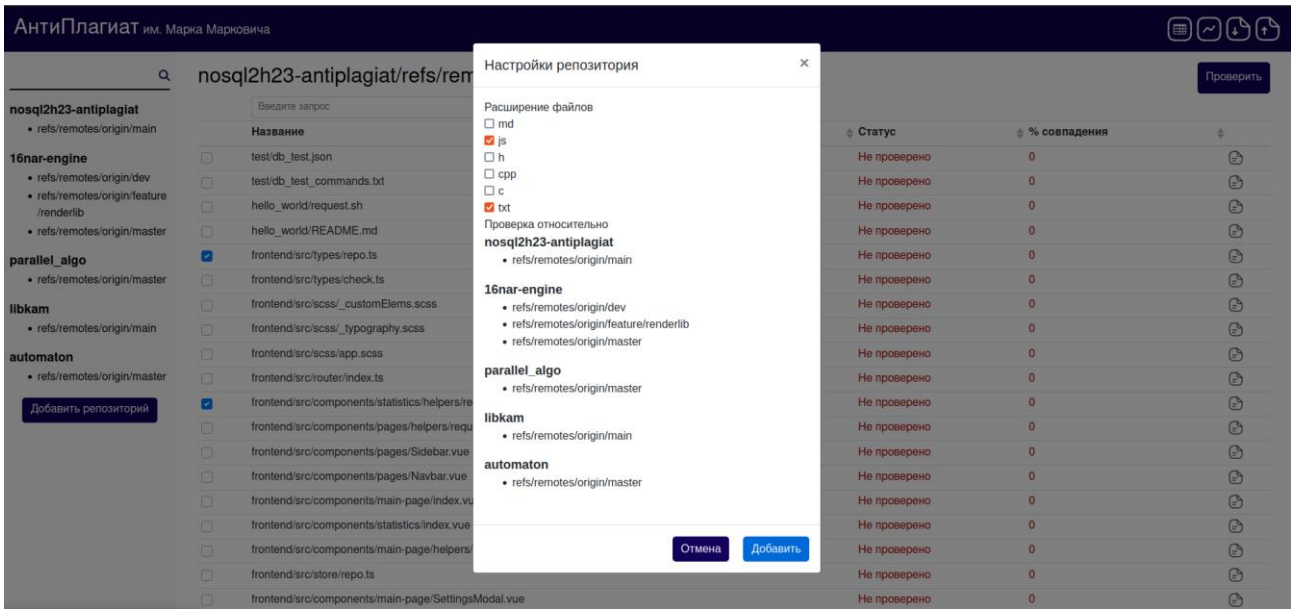

Рисунок 9. Экран настройки проверки выбранных файлов в репозитории.

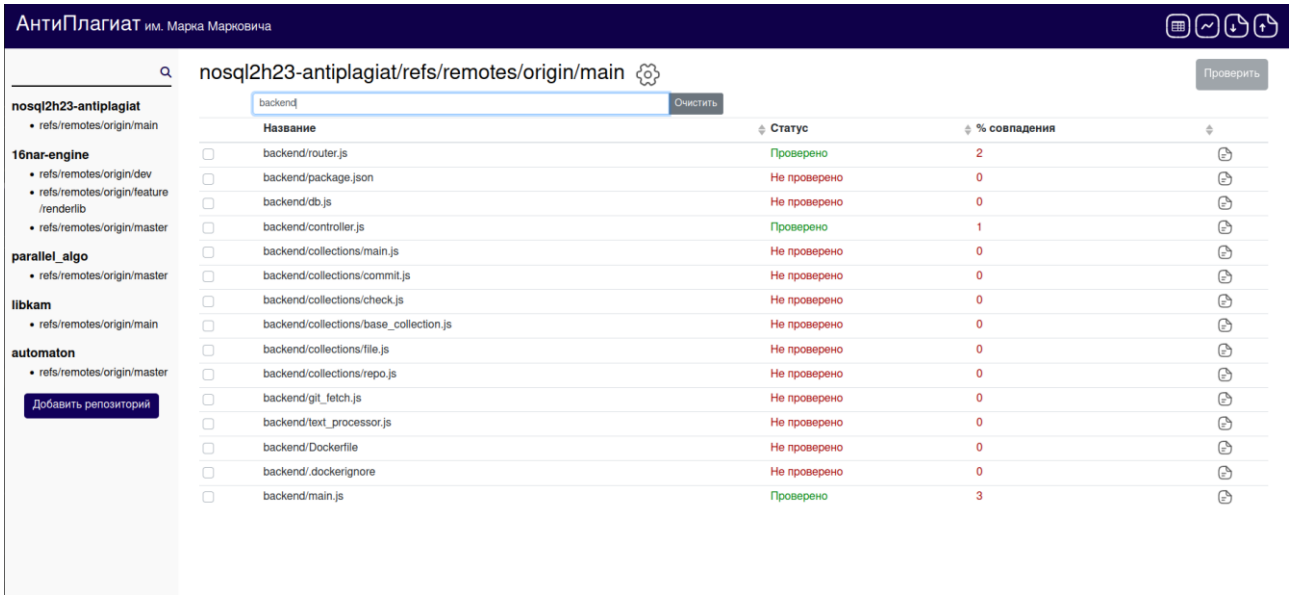

Рисунок 10. Экран с файлами репозитория после проверки,
файлы отфильтрованы по слову «backend».

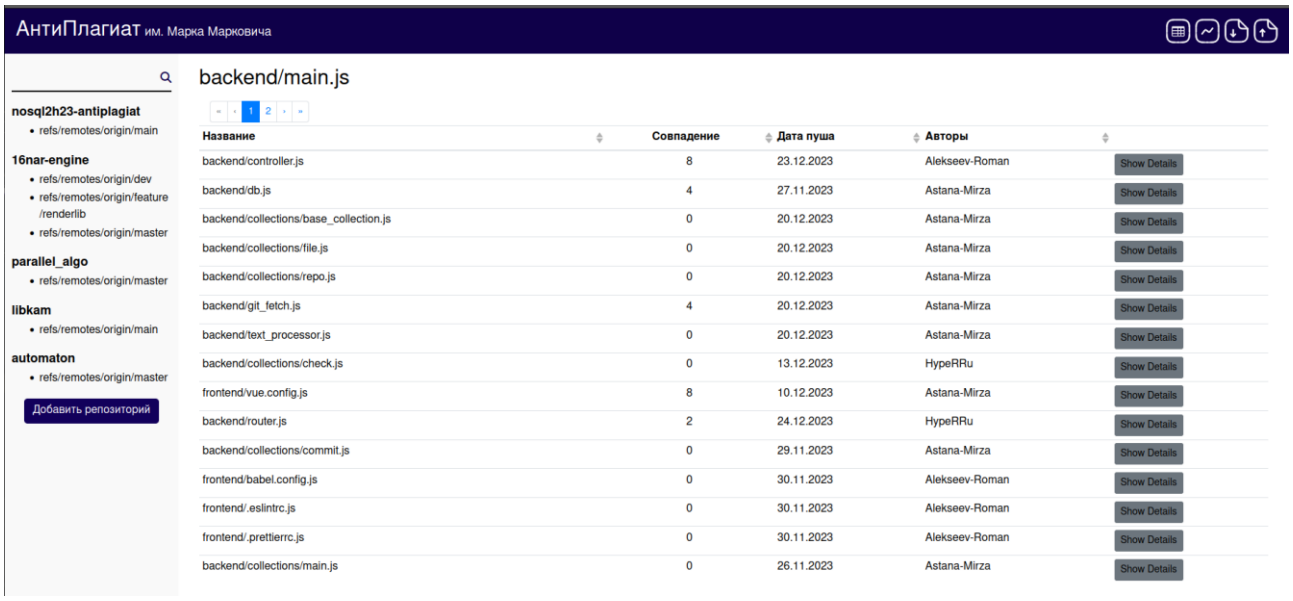

Рисунок 11. Экран с детальной информацией о проверке для файла «backend/main.js».

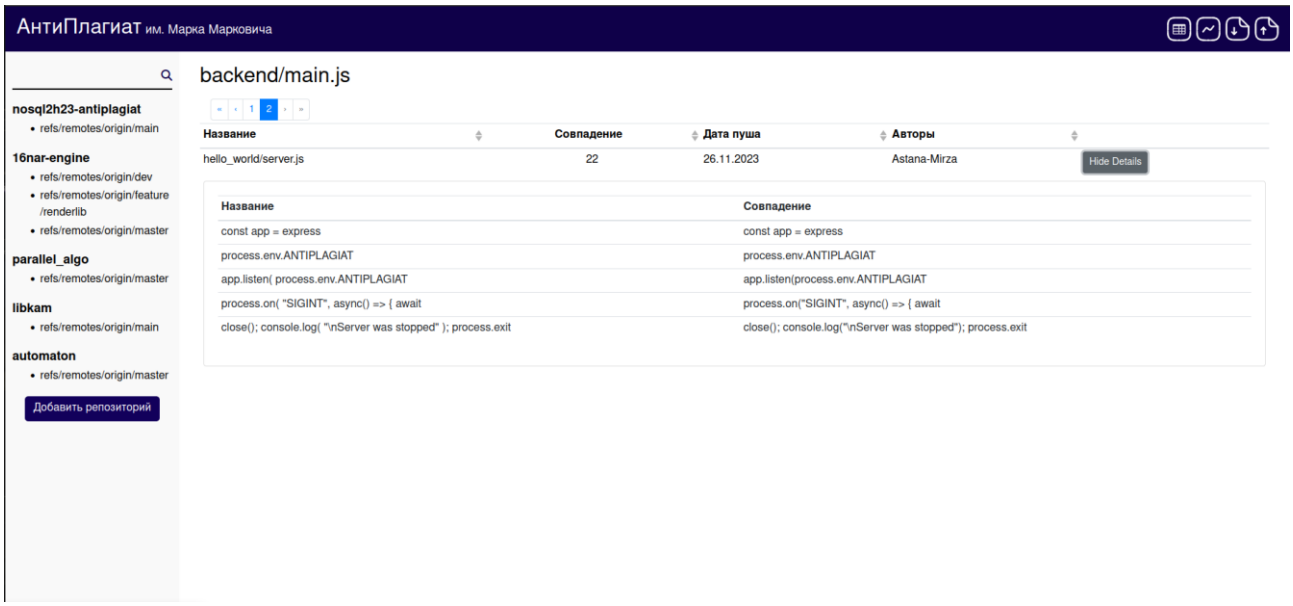

Рисунок 12. Экран со сравнением файлов «backend/main.js» и «hello_world/server.js».

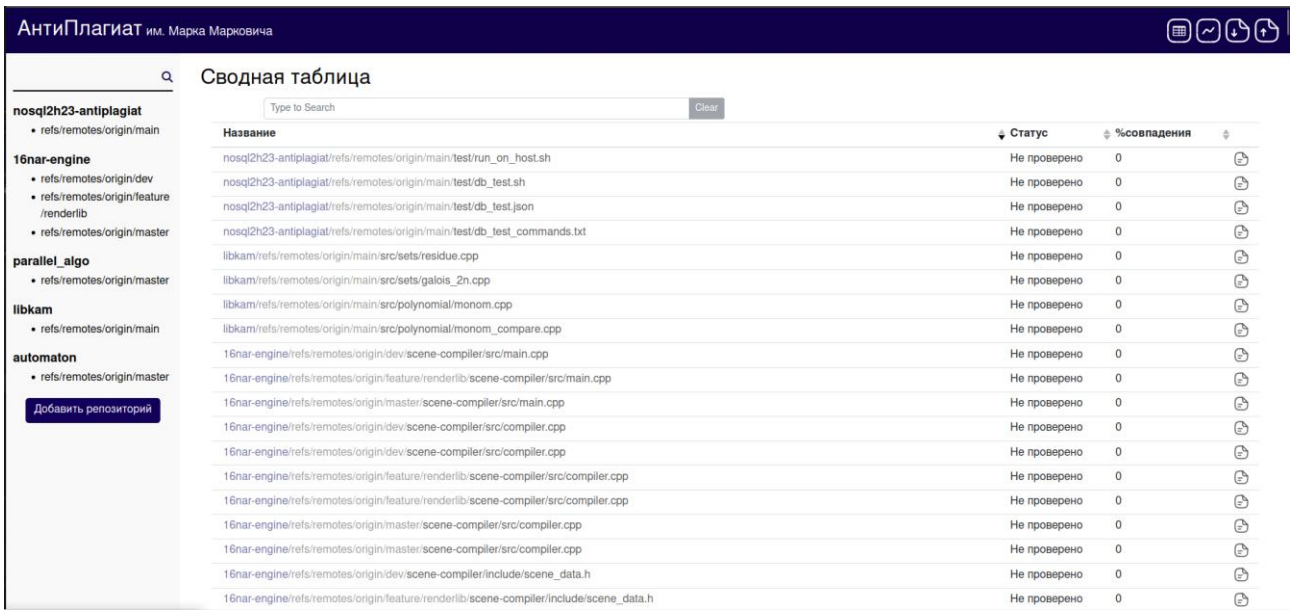

Рисунок 13. Сводная таблица со всеми файлами из всех репозиториев и веток.

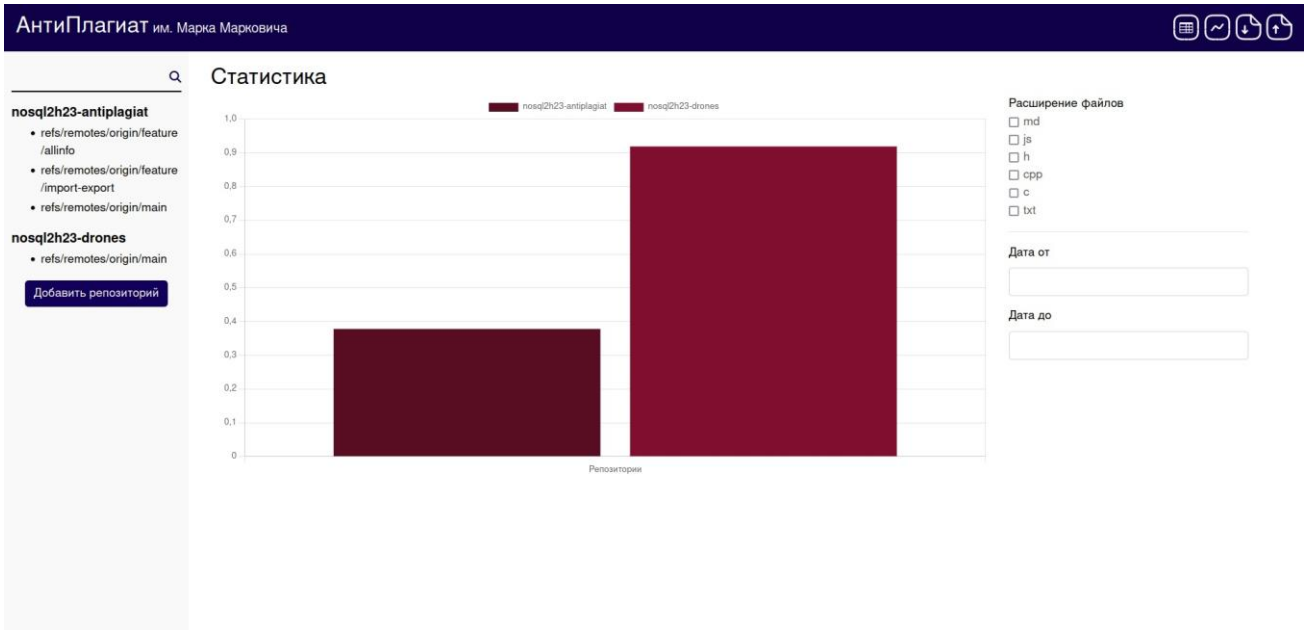

Рисунок 14. Экран с гистограммами статистики репозиториев.

ВЫВОДЫ

Достигнутые результаты.

В ходе работы разработан сервис локального плагиата (для нескольких репозиториев) для естественного языка. При помощи web-интерфейса пользователь может добавлять и настраивать репозитории, запускать проверки и просматривать их результаты и статистики.

Недостатки и пути для улучшения полученного решения.

В текущей реализации сервиса алгоритм проверки строго определен и не может быть настроен пользователем для более гибкого анализа на уровень плагиата. Также при анализе не учитываются падежи, склонения, множественное и единственное числа и т.д.

Одним из вариантов решения данных проблем является использование более сложных алгоритмов обработки естественного языка.

Будущее развитие решения.

Разработка приложения для ОС Windows 10 и 11, для ОС MacOS.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий проекта // GitHub. URL: https://github.com/moevm/nosql2h23-antiplagiat
2. Документация MongoDB // MongoDB.com. URL: https://www.mongodb.com/
3. Документация Vue.js // Vue.js. URL: https://ru.vuejs.org/

ПРИЛОЖЕНИЕ А

ДОКУМЕНТАЦИЯ ПО СБОРКЕ И РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ

1. Скачать проект из репозитория [1].
2. В директории проекта выполнить команду: *docker-compose up -- build*
3. Открыть web-приложение в браузере по адресу http://localhost:8080.

ПРИЛОЖЕНИЕ В

ИНСТРУКЦИЯ ДЛЯ ПОЛЬЗОВАТЕЛЯ

1. Добавление нового репозитория:
    a. Нажмите на кнопку «Добавить репозиторий»
    b. Вставьте ссылку на репозиторий
    c. Нажмите «Добавить»
2. Настройки репозитория:
    a. Нажмите на значок шестеренки
    b. Выберите интересующие вас настройки
    c. Нажмите «Ок»
3. Массовый экспорт:
    a. Нажмите на значок файла со стрелочкой вниз
4. Массовый импорт:
    a. Нажмите на значок файла со стрелочкой вверх
    b. Выберите файл формата JSON
5. Просмотр сводной таблицы:
    a. Нажмите на значок таблицы
6. Запуск проверки файлов:
    a. Выберите репозиторий в левой части экрана
    b. Отметьте интересующие вас файлы
    c. Нажмите на кнопку «Проверить»
    d. Дождитесь окончания проверки
7. Сортировка таблиц:
    a. Нажмите на название столбца, по которому хотите отсортировать. При одном нажатии таблица будет отсортирована по возрастанию, при двух по убыванию.