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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: Сервис-сборщик ответов на яндекс-формы и добавление их в таблицы

Студент гр. 1381 _____ Возмитель В.И.

Студент гр. 1384 _____ Степаненко Д.В.

Студент гр. 1381 _____ Тарасов К.О.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2024

ЗАДАНИЕ

Студенты

Возмитель В.И.

Степаненко Д.В.

Тарасов К.О.

Группы 1381, 1384

Тема работы: Разработка сервиса-сборщика ответов на яндекс-формы и добавление их в таблицы

Исходные данные:

Необходимо реализовать сервис для утилитарной задачи, а именно, автоматического импорта ответов из яндекс формы в яндекс таблицу.

Содержание пояснительной записки:

«Содержание», «Введение», «Сценарии использования», «Модель данных», «Разработанное приложение», «Заключение», «Список использованных источников», «Приложение»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 25.09.2024

Дата сдачи реферата: 24.12.2024

Дата защиты реферата: 24.12.2024

Студент гр. 1381 _____ Возмитель В.И.

Студент гр. 1384 _____ Степаненко Д.В.

Студент гр. 1381 _____ Тарасов К.О.

Преподаватель _____ Заславский М.М.

АННОТАЦИЯ

В данной работе рассматривается разработка сервиса для автоматического импорта ответов из Яндекс Форм в Яндекс Таблицы. Основной целью проекта является упрощение процесса интеграции данных, полученных от пользователей, в таблицы, что в настоящее время не поддерживается встроенными функциями Яндекс Форм. В работе описаны сценарии использования, модель данных, разработанное приложение и его архитектура. Для реализации сервиса использованы технологии MongoDB для хранения данных, Spring и Java для бэкенда, а также JavaScript и Vite для фронтенда. Результаты работы включают создание функционального интерфейса, возможность импорта и экспорта данных, а также средства для анализа статистики ответов.

SUMMARY

This paper discusses the development of a service for automatically importing responses from Yandex Forms to Yandex Tables. The main goal of the project is to simplify the process of integrating data received from users into tables, which is currently not supported by the built-in functions of Yandex Forms. The paper describes the usage scenarios, the data model, the developed application and it’s architecture. MongoDB technologies for data storage, Spring and Java technologies for the backend, as well as JavaScript and Vite technologies for the front-end are used to implement the service. The results of the work include the creation of a functional interface, the ability to import and export data, as well as tools for analyzing response statistics.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Пользователи сервиса Яндекс Форм сталкиваются с необходимостью интеграции данных, полученных из опросов, в таблицы. Существующие решения, такие как Google Формы, предлагают встроенные функции для автоматического импорта данных в таблицы, что значительно упрощает работу пользователей. Однако аналогичные возможности для Яндекс Форм отсутствуют, что создает трудности для пользователей.

Необходимо создать сервис для выполнения автоматического импорта ответов из ответов из Яндекс Форм в таблицы. Сервис должен обеспечивать синхронизацию произвольного количества пар «форма - таблица», ведение логов действий, администрирование пар в системе.

Предлагается создать сервис с использование технологий: MongoDB – для хранения данных опросов, таблиц и метаданных, Spring и Java – backend, JavaScript и Vite – frontend.

Качественные требования включают в себя создание таблиц ответов по привязанной форме, удобный пользовательский интерфейс, возможность импорта и экспорта данных, средства анализа статистики ответов.

1. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1.1. Макет UI

Для согласованной разработки сервиса был разработан макет пользовательского интерфейса (см. рис. 1).

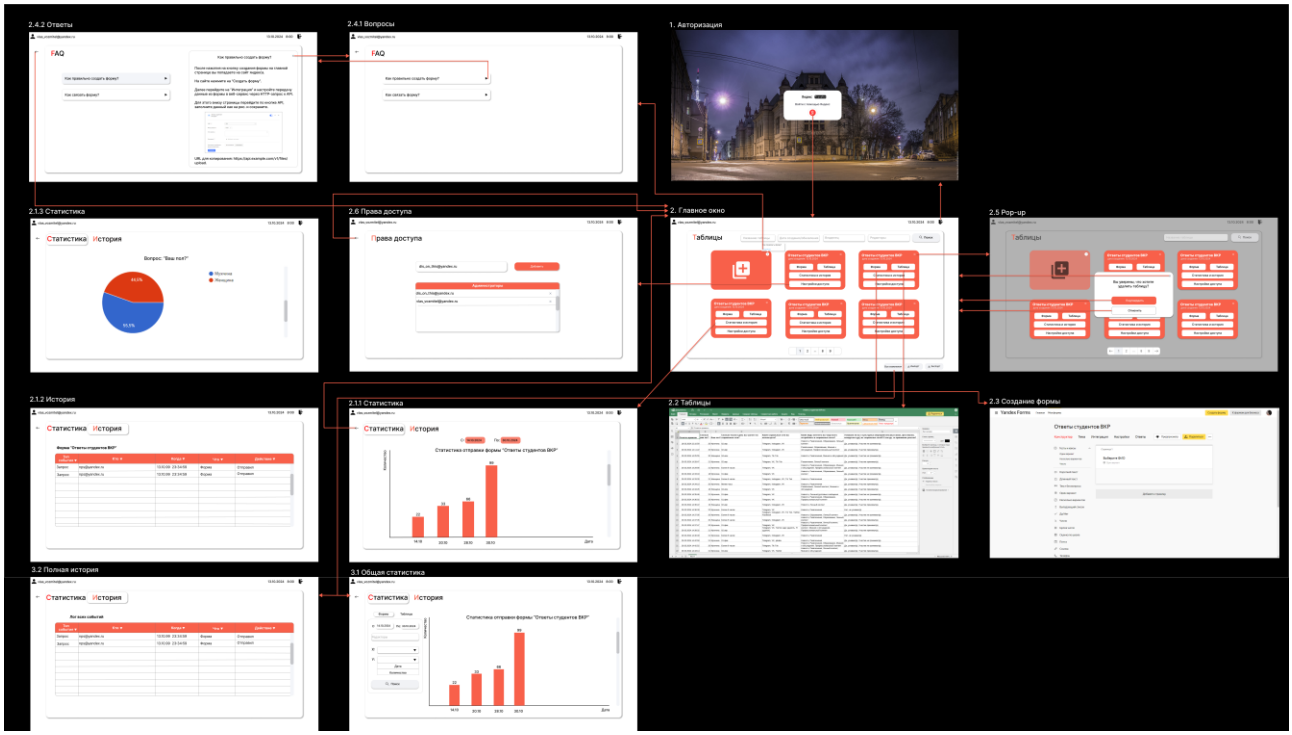

Рисунок 1 - Макет UI

1.2. Сценарии использования для задач

1. импорта данных: пользователь загружает данные в программу массово по кнопке «импорт» в главном меню из файла в формате json.
2. представления данных: для пользователя данные отображаются в виде таблиц. На главной стране отображены все связки таблица-форма с характеристиками: название формы, владелец формы, дата создания, редакторы и таблица. По нажатию на кнопку из столбца «Таблица» пользователь может посмотреть ответы, пришедшие на форму, включая ID ответа и время его создания. Все таблицы и их содержимое можно отфильтровать по дате создания, названию формы, владельцу, редакторам или ID ответа в верхних поисковых окнах.

3. анализа данных: для просмотра статистики отправки ответов пользователь должен нажать на кнопку «Общая статистика» в главном окне или кнопку «Статистика» в окне пары таблица-форма.
4. экспорта данных: пользователь может экспортировать все данные таблиц (ответы на формы, метаданные, связки формы-таблица) по нажатию кнопки «Экспорт» на главной странице. Далее все данные выгружаются в формате json.

1.3. Вывод об операциях

В системе будут преобладать операции чтения данных, так как основной функционал направлен на просмотр ответов, единожды пришедших с формы по POST-запросу, анализ статистики ответов, экспорта данных. Операции записи будут использоваться реже: для записи ответов в таблицу, импорта данных, редактирования доступа к таблицам.

2. МОДЕЛЬ ДАННЫХ

2.1. Нереляционная модель данных

Графическое представление в виде ER-диаграммы можно посмотреть на рисунке 2.

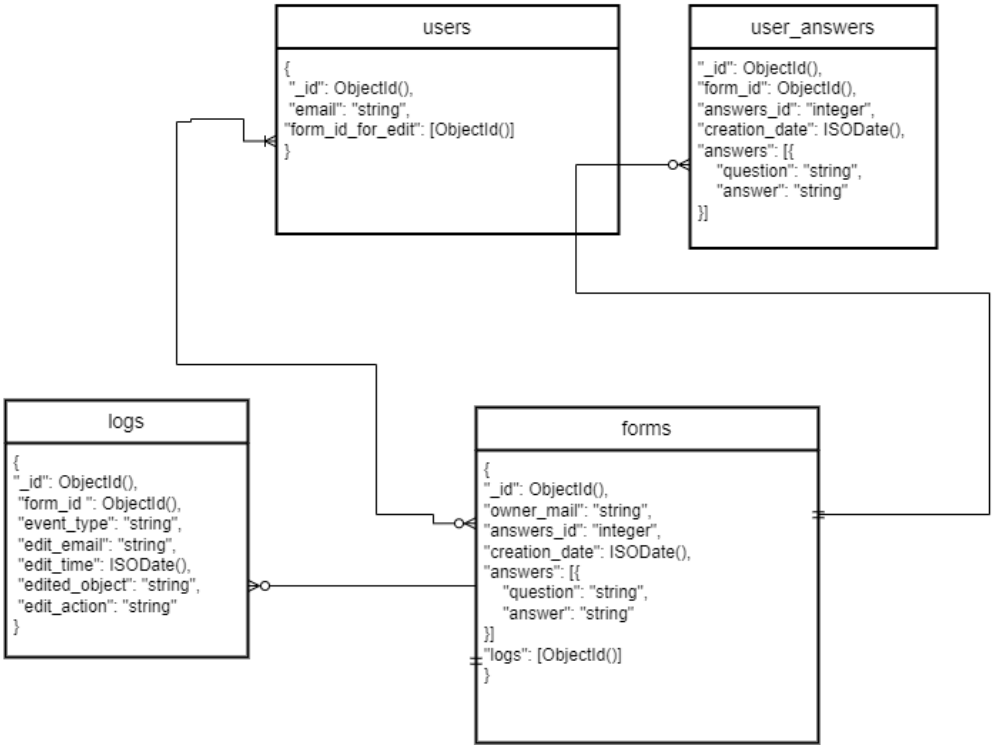

Рисунок 2 – Графическое представление ER-диаграммы для нереляционной модели данных

Всего в моделе данных 4 коллекции: *forms, users, tables, logs*. Первая хранит в себе уникальный идентификатор пользователя, почту пользователя, идентификатор кортежа ответов (от Яндекса), дату ответа на форму, кортеж ответов (вопрос, ответ), историю. Коллекция «*users*» хранит: уникальный идентификатор пользователя, почту пользователя, уникальный идентификатор формы, которую пользователь (user) может изменять, ID формы, атрибут для указания роли пользователя. Коллекция «*tables*» хранит: уникальный идентификатор кортежа ответов, уникальный идентификатор формы, идентификатор кортежа ответов (от Яндекса), дату ответа на форму, кортеж из вопросов и ответов. Коллекция «*logs*» хранит: уникальный идентификатор логов, уникальный идентификатор формы, тип лога, идентификатор

пользователя (почта), время последнего изменения, тип действия (трансляция, из вне, в сервисе, экспорт, импорт).

1) Коллекция *forms*:

- *_id* - уникальный идентификатор пользователя
- *owner_mail* - почта пользователя
- *answers_id* - идентификатор кортежа ответов (от сервиса Яндекса)
- *creation_date* - дата ответа на форму
- *question* - вопрос в форме
- *answer* – ответ на вопрос
- *logs* – история

2) Коллекция *users*:

- *_id* - уникальный идентификатор пользователя
- *email* - почта пользователя
- *form_id_for_edit* - уникальный идентификатор формы, которую пользователь может изменять
- *form_id* - ID формы
- *user_attribute* - атрибут для указания роли

3) Коллекция *tables*:

- _id - уникальный идентификатор кортежа ответов
- *form_id* - ID формы
- *answers_id* - идентификатор кортежа ответов (от сервиса Яндекса)
- *creation_date* - дата ответа на форму
- *question* - вопрос
- *answer* - ответ

4) Коллекция *logs*:

- *_id* - уникальный идентификатор логов
- *form_id* - уникальный идентификатор формы
- *event_type* - тип лога
- *user_email* - идентификатор пользователя (почта)
- *edit_time* – время изменения
- *edit_action* - действия (в зависимости от типа лога: трансляция, из вне, в сервисе, экспорт/импорт)

Предположительный объем данных при использовании базы данных MONGODB:

- В среднем на одной форме вопросов - 10 (Q)
- В среднем у формы редакторов - 2 (N)
- В среднем проходят опрос человек - 20 (H)
- В среднем количество форм - 3 (F)

Коллекция *forms*:

- *_id (ObjectId)*: 12 байт
- *owner_mail*: 320 байт
- [{*answers_id*: 4 байт
- *creation_date*: 8 байт
- *answers*: [{
- *question*: 1500 байт
- *answer*: 1000 байт}]}]
- *logs*: 12 байт * (H+5) // в среднем

Итого для формы: (12 + 320 + (4 + 8 + (1500 + 1000) * Q) * H) * F = 1501716 байта

Коллекция *users*:

- *_id (ObjectId)*: 12 байт
- *mail*: 320 байт
- *form_id_for_edit*: по 12 байт на каждую форму

Итого для пользователя: 12 + 320 + 12 * N = 356 байта

Коллекция *logs*:

- *_id (ObjectId)*: 12 байт
- *form_id (ObjectId):* 12 байт
- *event_type*: 50 байт
- *edit_email*: 320 байт
- *edit_time*: 8 байт
- *edit_action*: 100 байт

Итого для логов: (320 + 100 + 50 + 12 + 12 + 8) * (H + 5) = 12550 байта

Коллекция *tables*:

- *_id (ObjectId)*: 12 байт
- *form_id (ObjectId)*: 12 байт
- *answers_id*: 8 байт
- *creation_date*: 8 байт
- *answers*: [
- *question*: 1500 байт
- *answer*: 1000 байт]

Итого: (12 + 12 + 8 + (1500 + 1000) * Q) * H = 500640 байт

Общий итог по базе данных: (F * (500640 + 12550) + 356 + 1501716) * P = 3041642 * P

В модели присутствуют ссылающие на другие коллекции *id*, а также дублируемое хранение обычных данных:

1. В каждом объекте коллекции хранится ее собственный *id*
2. В коллекции *forms* хранится продублированный массив ответов для более быстрого доступа
3. *forms* хранит массив *_id* на *logs*, *logs* хранит *_id* таблицы с которой происходило взаимодействие
4. Форма ссылается на пользователя (владельца формы).

Итак, избыточность рассчитывается как: (1540886 * P)/(320 + 320 * 2 + (478 * 25 + (16 + 2500 * 10) * 20)*3) * P = 3041642/1537770 = 1.9779

Рост модели зависит от параметров:

1. Q - количество вопросов в форме (линейная)
2. N - количество редакторов у формы (линейная)
3. H - количество пройденный опросов (квадратичная: порождает логи)
4. F - количество форм у одного владельца (кубическая зависимость: форма порождает ответы на форму, логи)

Запросы к модели, с помощью которых реализуются сценарии использования перечислены далее:

1. Просмотр доступных таблиц:

```
db.forms.find({owner_mail: "owner_mail"})
```

1 запрос, 1 задействованная коллекция

2. История изменений:

```
db.logs.find({form_id: id})
```

1 запрос, 1 задействованная коллекция

3. Предоставить права доступа:

```
db.users.updateOne(
    {
```

```
        "email" : "tarasov@yandex.ru"
    },
    {
        $push: {
            "form_id_for_edit":
                "671d537bfade9f88c237ada4"
        }
    })
```

1 запрос, 1 задействованная коллекция

4. Импорт данных:

```
const data = fs.readFileSync('data.json', 'utf8');
const forms = JSON.parse(data);
const collection = db.collection('forms');
await collection.insertMany(forms);
```

1 запрос, 1 задействованная коллекция

5. Экспорт данных:

```
const collection = db.forms
const result = collection.aggregate([{$lookup:
        {
            from: "logs",
            localField: "logs",
            foreignField: "_id",
            as: "logs"
        }
}])
const doc = result.toArray()
const json = JSON.stringify(doc, null, 2)
fs.writeFileSync('data.csv', json, 'utf8');
```

1 запрос, 2 задействованных коллекции

6. Статистика по всем таблицам:

```
db.logs.find({})
```

1 запрос, 1 задействованная коллекция

7. Все формы, у которых не было ответов за месяц:

```
db.forms.find({
    $expr: {
        $gt: [
            {$subtract: ["$creation_date", new Date()]},
            30 * 24 * 60 * 60 * 1000
        ]
    }
```

```
})
```

1 запрос, 1 задействованная коллекция

8. Все ответы, у которых нет id формы

```
db.tables.find({form_id: null})
```

1 запрос, 1 задействованная коллекция

9. Все неопубликованные (в нужной таблице) ответы на форму:

```
db.user_answer.aggregate([
    {
        $lookup: {
            from: "logs",
            localField: "_id",
            foreignField: "edit_action",
            as: "log_info"
        }
    },
    {
        $unwind: "$log_info"
    },
    {
        $match: {// Фильтруем документы по условию
            "log_info.edit_type": "Warning",
            "log_info.user_email": @Searching_id1
        }
    }
```

1 запрос, 2 задействованных коллекции

2.2. Аналог модели данных для SQL СУБД

Графическое представление в виде ER-диаграммы можно посмотреть на рисунке 3.

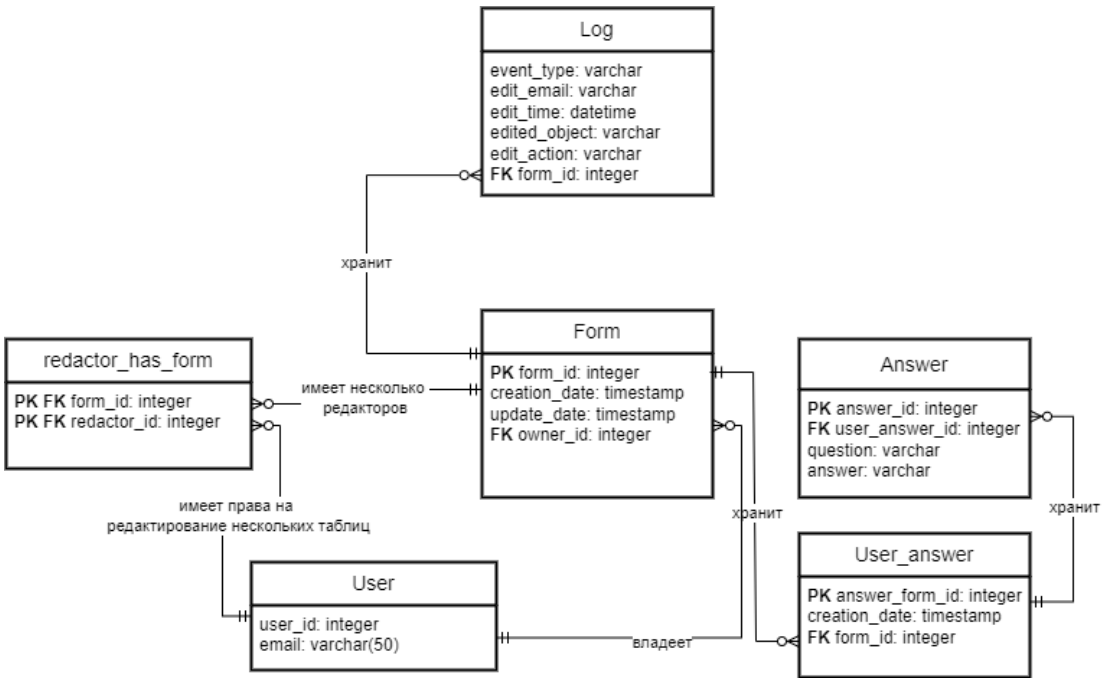

Рисунок 3 - Графическое представление ER-диаграммы для реляционной модели данных

1) Таблица *User*:

- Назначение: хранение информации о пользователе
- Идентификатор (*user_id*) - integer, 4 байта
- Почта(*email*) - varchar(50) - занимает 50 байт

2) Таблица *Form*:

- Назначение: хранение информации о форме, связь формы и ответов на неё, связь с пользователями
- Идентификатор (*form_id*) - integer, 4 байта
- Дата создания формы (*creation_date*) - timestamp занимает 8 байт
- Дата обновления формы (*update_date*) - занимает 8 байт
- Идентификатор создателя (*owner_id*) *integer*, занимает 4 байта

3) Таблица *redactor_has_form*:

- Назначение: таблица для связи многое ко многим - у формы может быть много редакторов, и у редакторов может быть много форм
- Идентификатор формы (*form_id*) - *integer*, 4 байта

- Идентификатор редактора (*redactor_id*) - *integer*, 4 байта

4) Таблица *Log*:

- Назначение: хранит все действия, которые совершались над таблицей (логи)
- Тип события, которое произошло (*event_type*) - *varchar*, в среднем 6 байт
- Почта пользователя, который совершил изменение (*edit_email*) - *varchar*(50), 50 байт
- Время произошедшего события (*edit_time*) - *timestamp*, в памяти занимает 8 байт
- Произошедшее событие (*edit_action*) - *varchar*, в среднем 20 байт
- Идентификатор формы (*form_id*) - *integer*, 4 байта

5) Таблица *Answer*:

- Назначение: хранит информацию об ответе на один из вопросов опроса
- Идентификатор (*answer_id*) - *integer*, 4 байта
- Идентификатор результата прохождения всего опроса (*user_answer_id*) - *integer*, 4 байта
- Заданный вопрос (*question*) - *varchar*, в среднем 30 байт
- Ответ на заданный вопрос (*answer*) - *varchar*, в среднем 30 байт

6) Таблица *Tables*:

- Назначение: хранит одно прохождение формы пользователем
- Идентификатор (*answer_form_id*) - integer, 4 байта
- Дата отправки ответа (*creation_date*) - *timestamp*, 8 байт
- Идентификатор формы (*form_id*) - *integer*, 4 байта

Вычисляем средний размер объектов каждой сущности:

- User: 4 + 50 = 54 байта

- Form: 4 + 8 + 8 + 4 = 24 байта
- redactor_has_form: 4 + 4 = 8 байт
- Log: 6 + 50 + 8 + 20 + 4 = 88 байт
- Answer: 4 + 4 + 30 + 30 = 68 байт
- Tables: 4 + 4 + 8 = 16 байт

Считаем, что у каждый формы в среднем 10 вопросов, количество пользователей нашего сервиса, к примеру - 3, редакторов у каждой формы - 2, в среднем 20 человек проходят опрос, возьмём за N количество форм, прошедших опрос:

$V(N)=N*((20*16)+(68*200)+88+24+(8*2))=14048*N$ байта

Избыточными данными у таблиц можно считать можно считать внешние ключи:

- Redactor_has_form: 8 байт
- Form: 4 байта
- Log: 4 байта
- Answer: 4 байта
- Tables: 4 байта

Вычисляем чистый объём данных: $V(N) = N * ((12*20) + (64*200) + 84 + 20 + 0)) = 13144 * N$ байта

Далее рассчитываем избыточность данных: $V(N)Vclean(N) = 14048 * 13144 * N \approx 1{,}06877662812$

Модель будет расти линейно при создании объекта Form. При этом очень важно количество вопросов и количество прошедших опрос людей, так как это сильно влияет на объём созданных данных. При создании объекта Form, также будет создаваться объект Log.

Запросы к модели, с помощью которых реализуются сценарии использования перечислены далее:

1. Просмотр доступных таблиц:

```
select * from Form where owner_id = @Searching_id;
```

1 запрос, 1 задействованная коллекция

2. История изменений:

```
select * from Log
where form_id = @Searching_id;
```

1 запрос, 1 задействованная коллекция

3. Предоставить права доступа:

```
insert into redactor_has_form(form_id, redactor_id) VALUES
(@Searching_id1, @Searching_id2);
```

1 запрос, 1 задействованная коллекция

4. Импорт данных:

```
copy Form(creation_date, update_date, owner_id) FROM 'dataset.csv'
DELIMITER ',' CSV HEADER;
```

1 запрос, 1 задействованная коллекция

5. Экспорт данных:

```
copy (select * from Form
    join "User" U on U.user_id = Form.owner_id
    join log l on Form.form_id = l.form_id
    join user_answer ua on Form.form_id = ua.form_id
    join answer a on ua.answer_form_id = a.answer_id
    ) TO 'dataset.csv' DELIMITER ',' CSV HEADER;
```

2 запроса, 5 задействованных коллекций

6. Статистика по всем таблицам:

```
select * from log where form_id in (@Searching_id's);
```

1 запрос, 1 задействованная коллекция

7. Все формы, у которых не было ответов за месяц:

```
select * from form where extract(DAY FROM (update_date -
current_timestamp)) > 30
```

1 запрос, 1 задействованная коллекция

8. Все ответы, у которых нет id формы:

```
select * from user_answer where form_id == null
```

1 запрос, 1 задействованная коллекция

9. Все неопубликованные (в нужной таблице) ответы на форму:

```
select * from user_answer where user_answer_id
 in (select edit_action from logs
 where edit_action == user_answer_id and edit_type == 'Warning' and
user_email == @Searching_id1
)
```

2 запроса, 2 задействованных коллекций

2.3. Сравнение моделей

Удельный объем информации в реляционной БД занимает меньше места, так как данные хранятся в нормализованной форме, без дублирования (за исключением id других таблиц), объем основной части хранящейся информации увеличивается линейно с ростом приходящих вопросов. В no-sql БД удельный объем занимает больше места, так как данные хранятся в де-нормализованной форме с дублированием для более быстрого доступа к ним. В этом случае при росте числа приходящих ответов, объем увеличивается с квадратной скоростью.

В postgreSQL почти для всех use case требуется 1 запрос, кроме экспорта данных - где запросов 2. В mongoDB во всех use case требуется 1 запрос. Количество задействованных коллекций также различается только в экспорте данных: postgreSQL - 5, mongoDB - 2.
Исходя из полученных значений, однозначных выводов сделать нельзя. Но можно сказать, что в среднем количество коллекций, задействованных в запрос, у MongoDb меньше.

2.4. Выводы

Подведем итоги. Реляционная БД поддерживает структурированные данные и строгий формат схемы, транзакции, обеспечивает компактность данных. По скорости обработки запросов - медленная (не подходит для хранения больших объемов неструктурированных данных)!

No - SQL модель: данные дублируются, что дает больше возможностей в работе с объемами больших данных (уменьшение времени обработки запросов - схема хранения данных более простая, сложность доступа к данным всегда константная). Транзакционность в данном случае не играет большой роли. Коэффициент избыточности = 1.9779, что повышает требования к хранилищу данных почти в 2 раза!

Таким образом, можно сделать вывод, что для нашего приложения более подходящей будет схема с MongoDb. Несмотря на большой недостаток с памятью, мы получаем высокое быстродействие - которое необходимо пользователям для мгновенной трансляции ответов форм в яндекс-таблицу.

3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

Система построена на микросервисной архитектуре и состоит из следующих модулей:

1. Frontend модуль включает в себя:

Web-интерфейс на JavaScript;

Визуализацию данных;

Взаимодействие с пользователями;

Реализует все описанные пользовательские сценарии.

2. Backend модуль:

API-сервер для обработки запросов;

Бизнес-логика приложения;

Управление авторизацией;

Обработка импорта/экспорта данных.

3. Модуль хранения данных:

MongoDB для работы с информацией, хранящейся в формах и таблицах, метаданными (информация о пользователях, формах, связках таблица-форма).

Использованные технологии:

Frontend: JS, Vite

Backend: Java, Spring boot

Базы данных: MongoDB

Контейнеризация: Docker

API: REST

Примеры работы приложена можно посмотреть на рисунках 4-10.

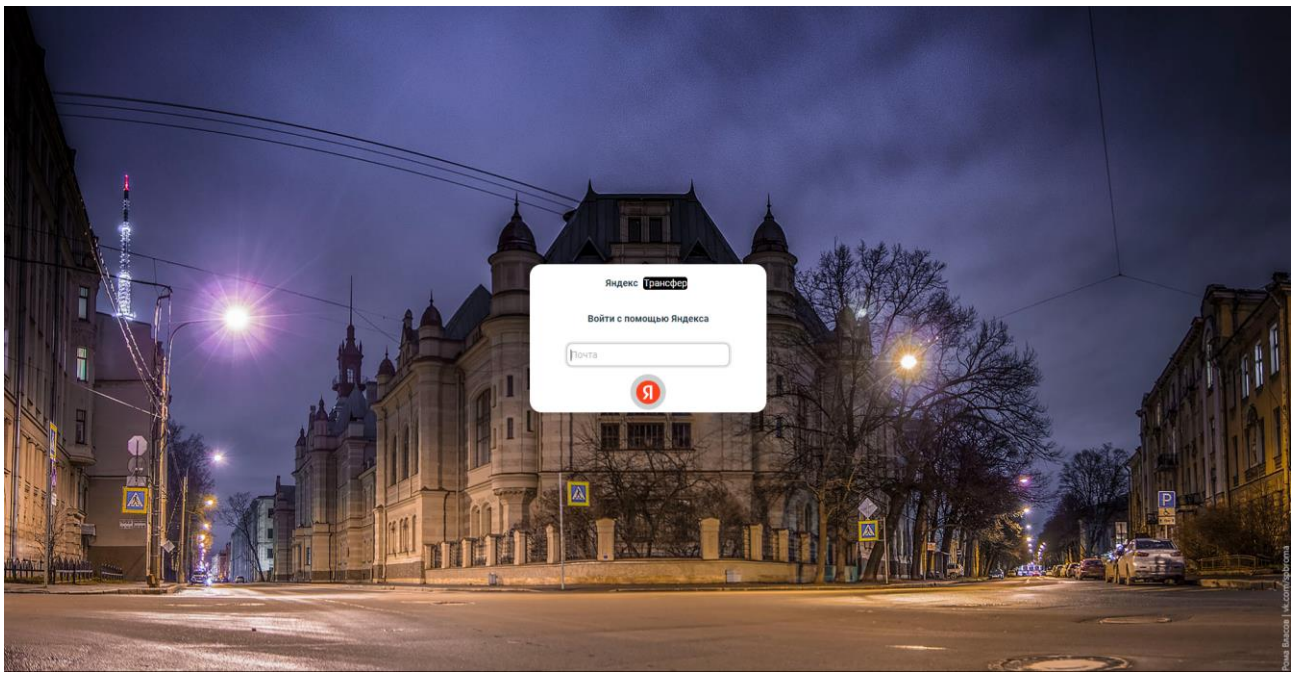

Рисунок 4 – Окно авторизации

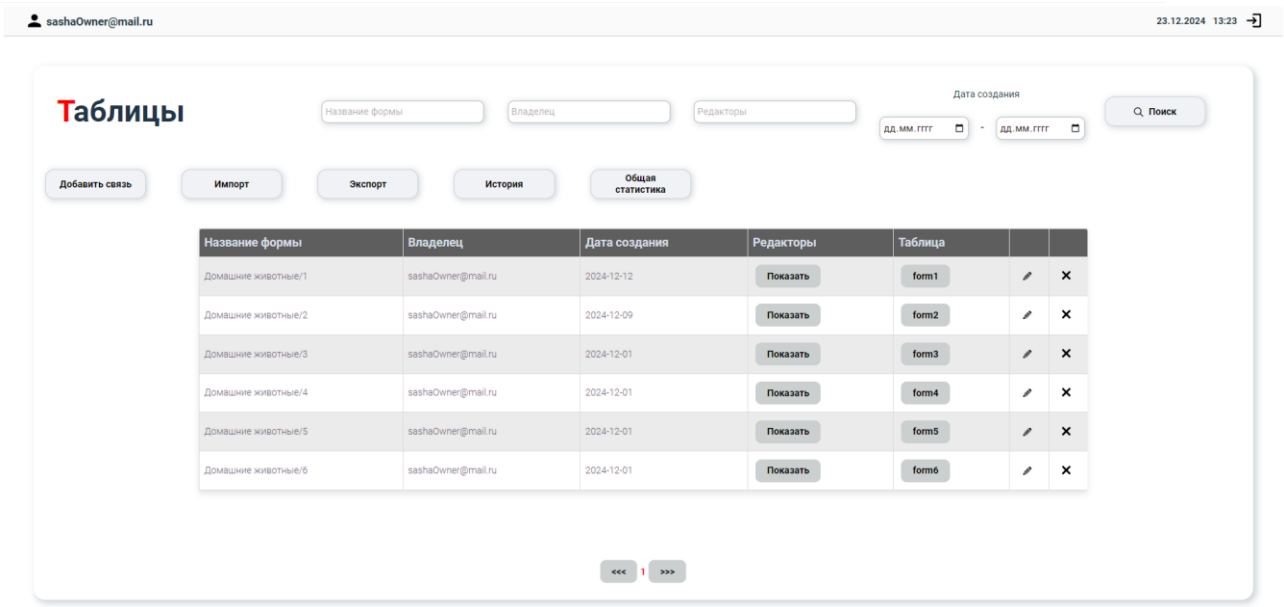

Рисунок 5 – Главное окно

Рисунок 6 – Окно добавление свяхки таблица-форма

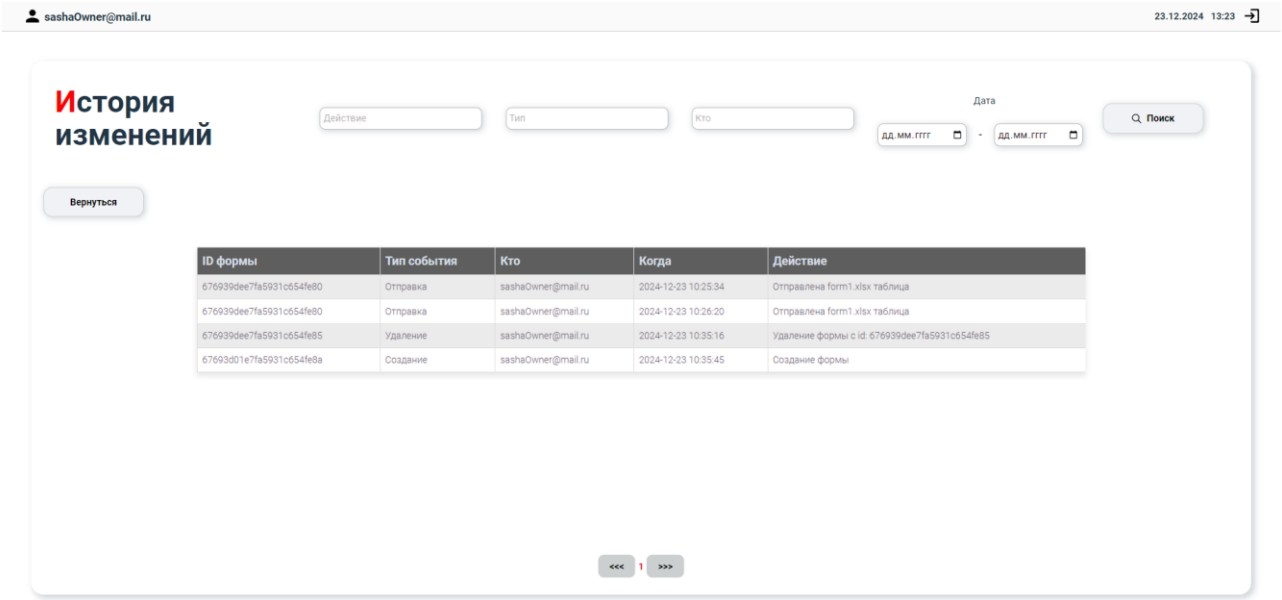

Рисунок 7 – Окно истории изменений (прросмотр логов)

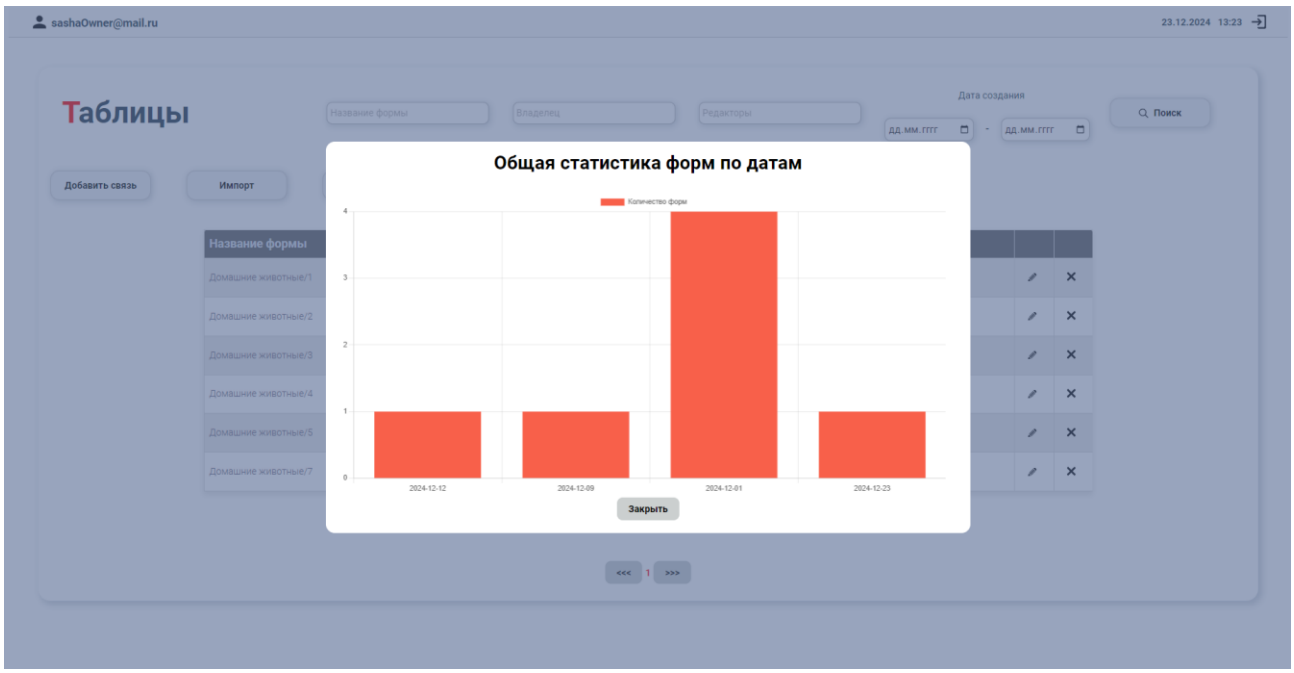

Рисунок 8 – Окно общей статистики отправки ответов

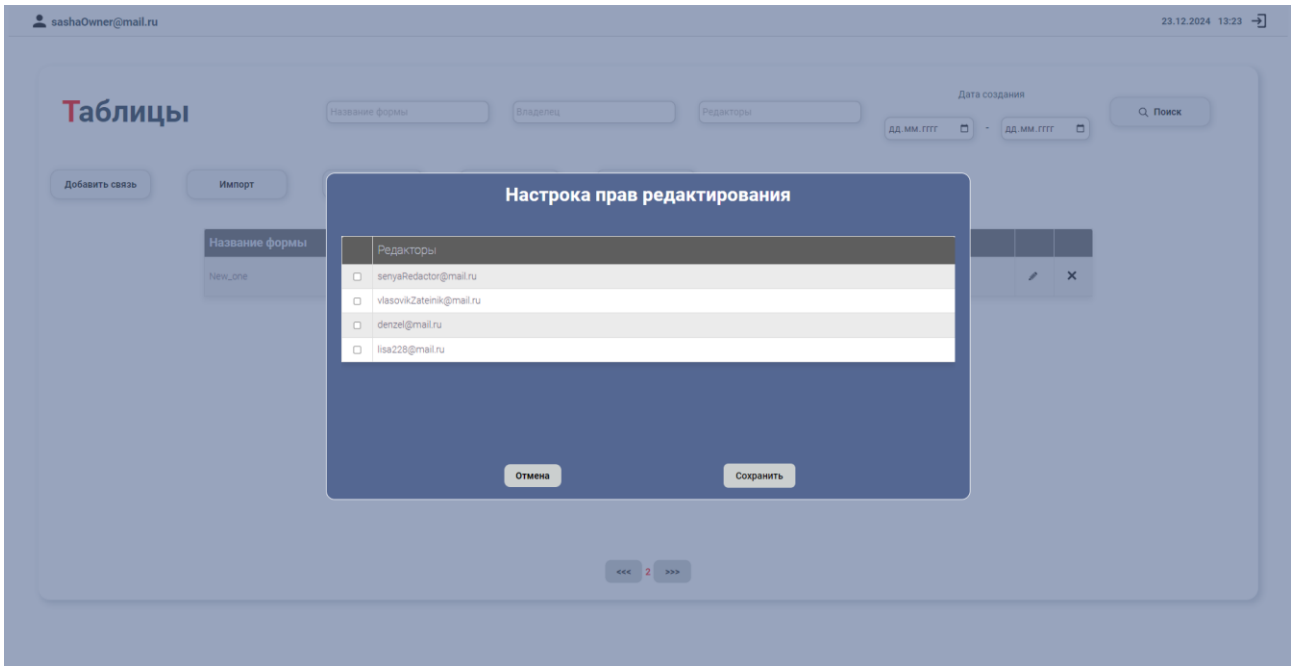

Рисунок 9 – Окно настройки прав доступа к связки таблица-форма

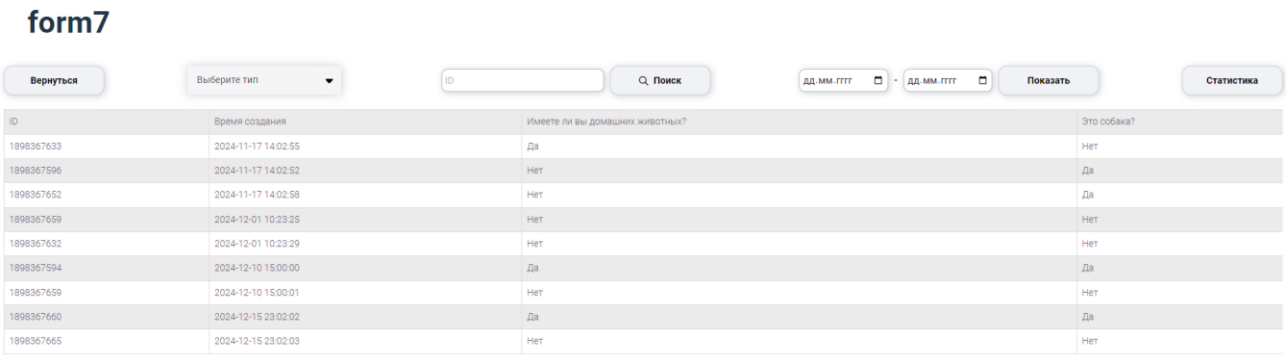

Рисунок 10 – Окно связки форма-таблица

ЗАКЛЮЧЕНИЕ

Достигнутые результаты.

В ходе работы реализовано приложение для автоматического импорта ответов из Яндекс Форм в Яндекс Таблицы, которое позволяет пользователям эффективно интегрировать данные, полученные от опросов, в таблицы. Пользователи могут просматривать ответы, фильтровать их по различным критериям, а также анализировать статистику. Приложение также предоставляет функции импорта и экспорта данных, что значительно упрощает работу с информацией.

Если говорить о выбранном решении для хранения данных, то на основе проведенного анализа можно сделать вывод: что использование нереляционной модели данных на базе MongoDB обеспечивает большую гибкость и простоту в добавлении новых объектов, что делает её более подходящей для большого объема динамически изменяющихся данных, несмотря на увеличения объема хранимой информации в 2 раза по сравнению с реляционными БД.

Недостатки и пути для улучшения полученного решения.

На данный момент в приложении отсутствует возможность регистрации через Яндекс почту. Также не хватает расширенных инструментов для анализа данных, что ограничивает возможности пользователей в получении статистики.

Будущее развитие решения.

Планируется внедрение системы уведомлений для пользователей о статусе импорта данных и возможных ошибках, а также интеграция с другими сервисами для расширения функциональности, например, с API других платформ для автоматизации процессов.

ПРИЛОЖЕНИЕ

Запуск приложения.

1. Клонировать репозиторий и запустить:
   git clone https://github.com/moevm/nosql2h24-yandex.git
   docker compose -f ./docker-compose.yml build
   docker compose -f ./docker-compose.yml up
2. Открыть в браузере:
   http://localhost:3000/
3. Войти в систему:
   Логин: sashaOwner@mail.ru

Функционал.

- Создание и удаление связок таблица-форма
- Сбор ответов, приходящих на форму
- Отображение ответов в таблицах
- Фильтрация данных
- Отображение статистики использования сервиса
- Экспорт и импорт всех данных приложения

Требования.

- Docker
- Ubuntu 22.04+
- Свободные порты 3000 и 8080

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Data model design - MongoDB 3.4 manual // www.mongodb.com URL: https://www.mongodb.com/docs/manual/core/data-model-design/ (дата обращения: 22.12.2024).
2. Chart.js // chartjs.org URL: https://spring.io/projects/spring-boot (дата обращения: 23.12.2024).
3. Spring Boot // spring.io URL: https://spring.io/projects/spring-boot (дата обращения: 23.12.2024).
4. Service of integration{target App} with {source App} // albato.com URL: https://albato.com/ (дата обращения: 23.12.2024).
5. Docker Hub // hub.docker.com URL: https://hub.docker.com/ (дата обращения: 23.12.2024).