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: Построение графа ссылок на web-странице.

Студент гр. 0382 _____ Корсунов А.А.

Студент гр. 0382 _____ Самулевич В.А.

Студент гр. 0382 _____ Гудов Н.Р.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2023

ЗАДАНИЕ

Студенты

Корсунов А.А.

Самулевич В.А.

Гудов Н.Р.

Группа 0382

Тема проекта: Построение графа ссылок на web-странице.

Исходные данные:

Необходимо реализовать приложение построения графа ссылок на web-странице с помощью СУБД Neo4j.

Содержание пояснительной записки:

«Содержание»

«Введение»

«Сценарий использования»

«Модель данных»

«Разработанное приложение»

«Выводы»

«Приложения»

«Литература»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 20.09.2023

Дата сдачи реферата: 23.12.2023

Дата защиты реферата: 23.12.2023

АННОТАЦИЯ

В рамках данного курса предполагалось разработать какое-либо приложение в команде на одну из поставленных тем. Была выбрана тема создания приложения построения графа ссылок на web-странице с помощью СУБД Neo4j. Во внимание будут приниматься такие аспекты как производительность и удобство разработки. Найти исходный код и всю дополнительную информацию можно по ссылке: https://github.com/moevm/nosql2h23-web-graph

ANNOTATION

As part of this course, it was supposed to develop an application in a team on one of the given topics. The topic of creating an application for constructing a graph of links on a web page using the Neo4j DBMS was chosen. Aspects such as performance and ease of development will be taken into account. You can find the source code and all additional information at the link: https://github.com/moevm/nosql2h23-web-graph

1. ВВЕДЕНИЕ

1.1 Актуальность решаемой проблемы

Цель работы - создать веб-приложения, предоставляющее возможность построения графа ссылок на web-страницах.

1.2 Постановка задачи

Полученное приложение должно решать следующие задачи:

- Производить построение графа и таблицы ссылок, по начальной ссылке, заданной пользователем.
- Предоставлять дополнительный функционал, по обработке и извлечению информации уже построенного графа.

1.3 Предлагаемое решение

Для решения поставленной задачи необходимо разработать web приложение, решающее задачи из предыдущего пункта.

1.4 Качественные требования к решению

Веб-приложение должно быть разработано с использованием СУБД neo4j и обладать интерфейсом, понятным для пользователя.

2. СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯ

2.1 Макет UI

Рис.1 – Начальное окно приложения.

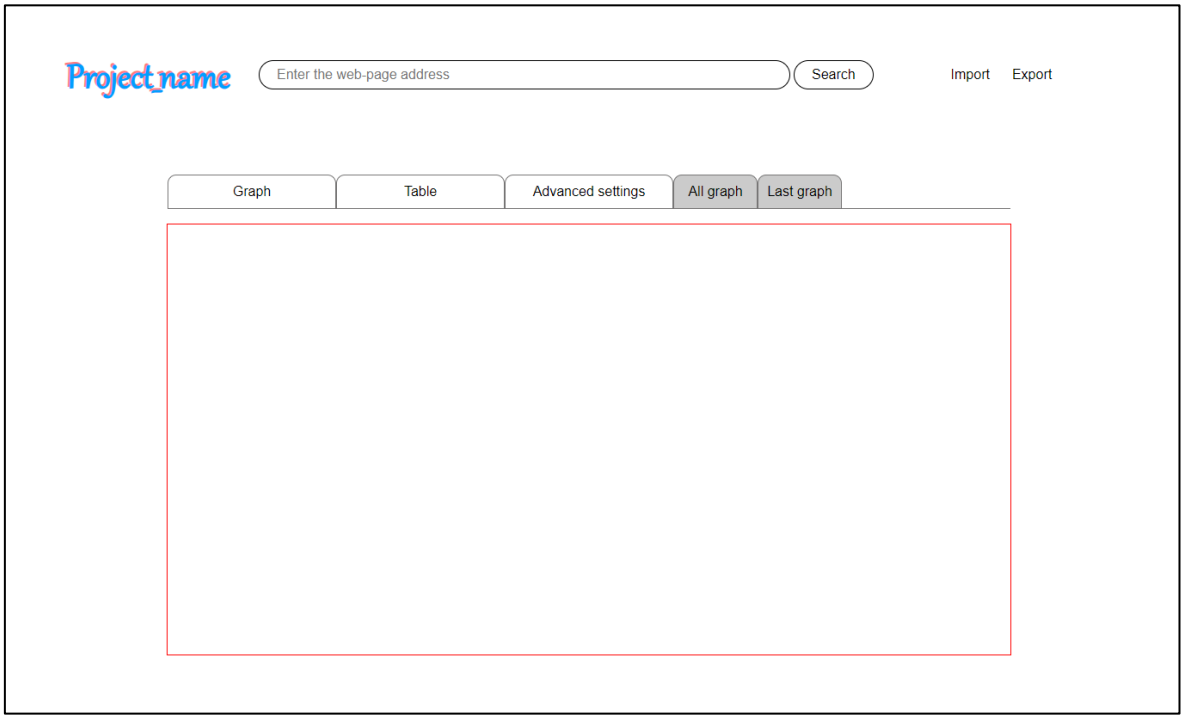


Рис.2 – Вкладка “Граф”.

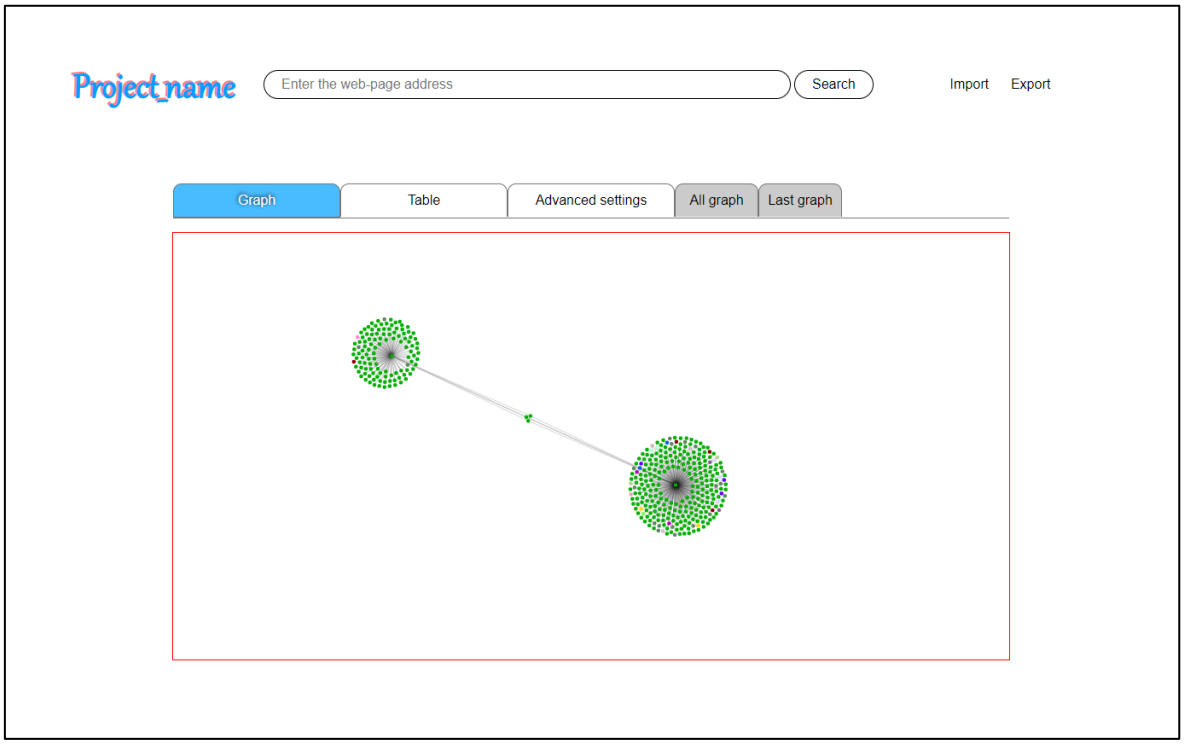


Рис.3 – Вкладка “Таблица”.

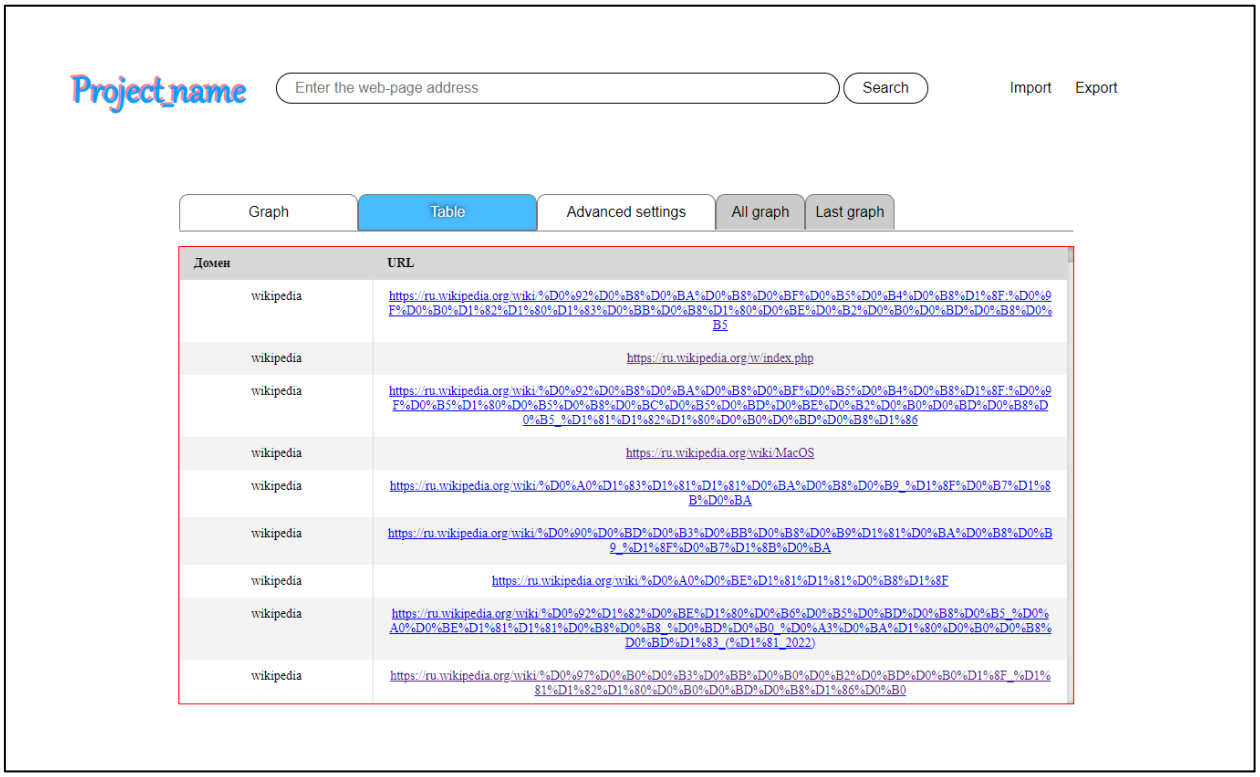


Рис.4 – Вкладка “Дополнительных параметров”.

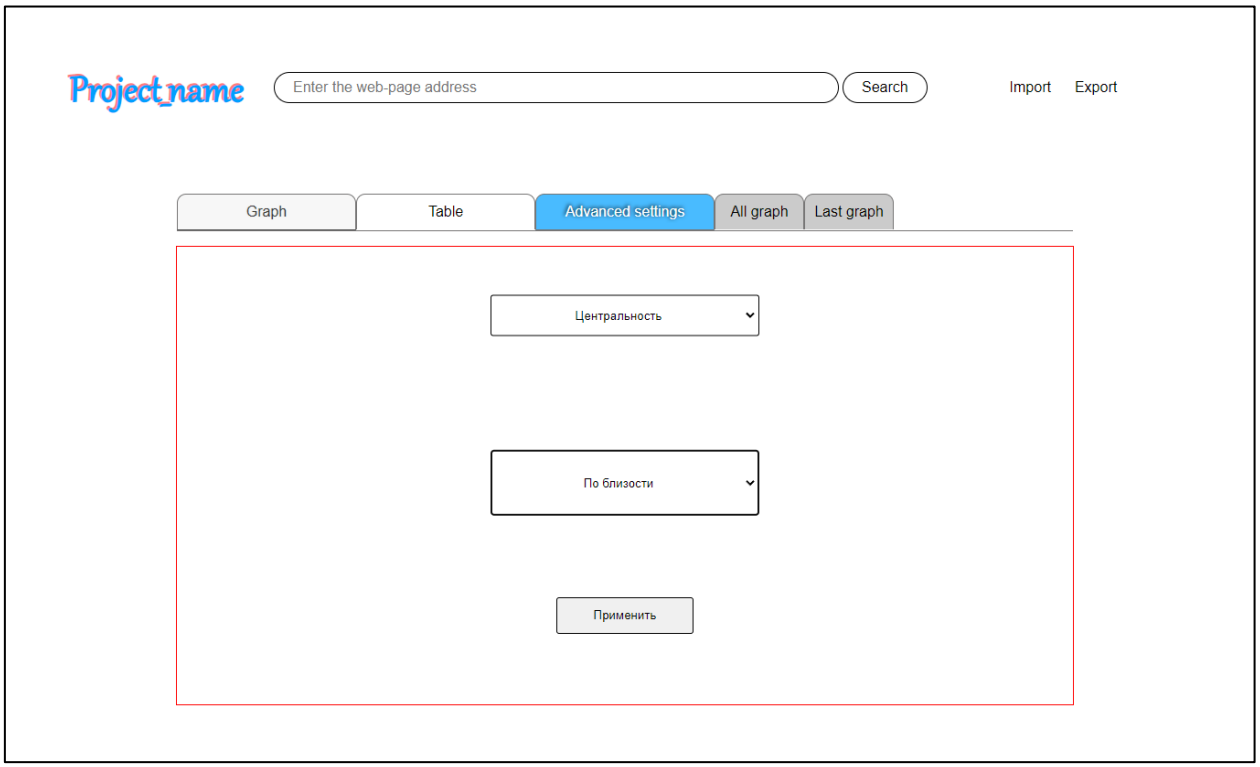


Рис.5 – Кнопки на странице.

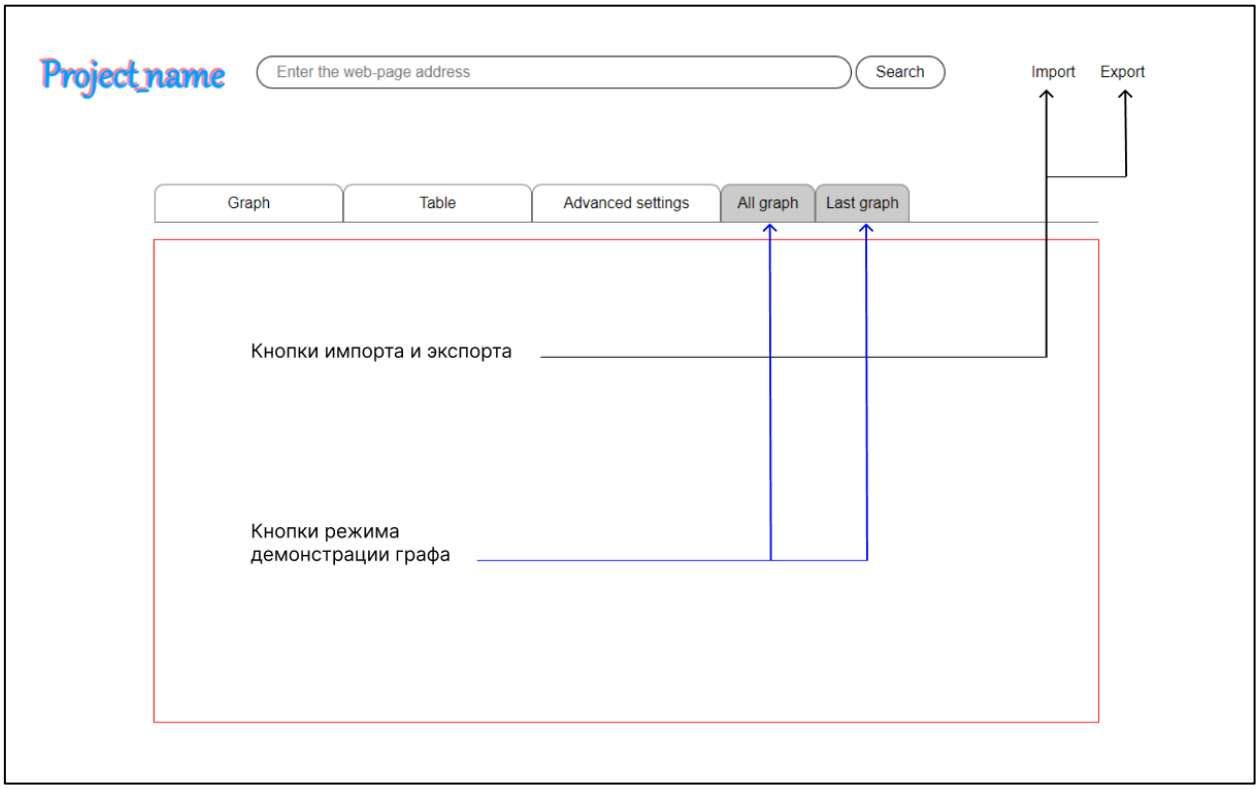
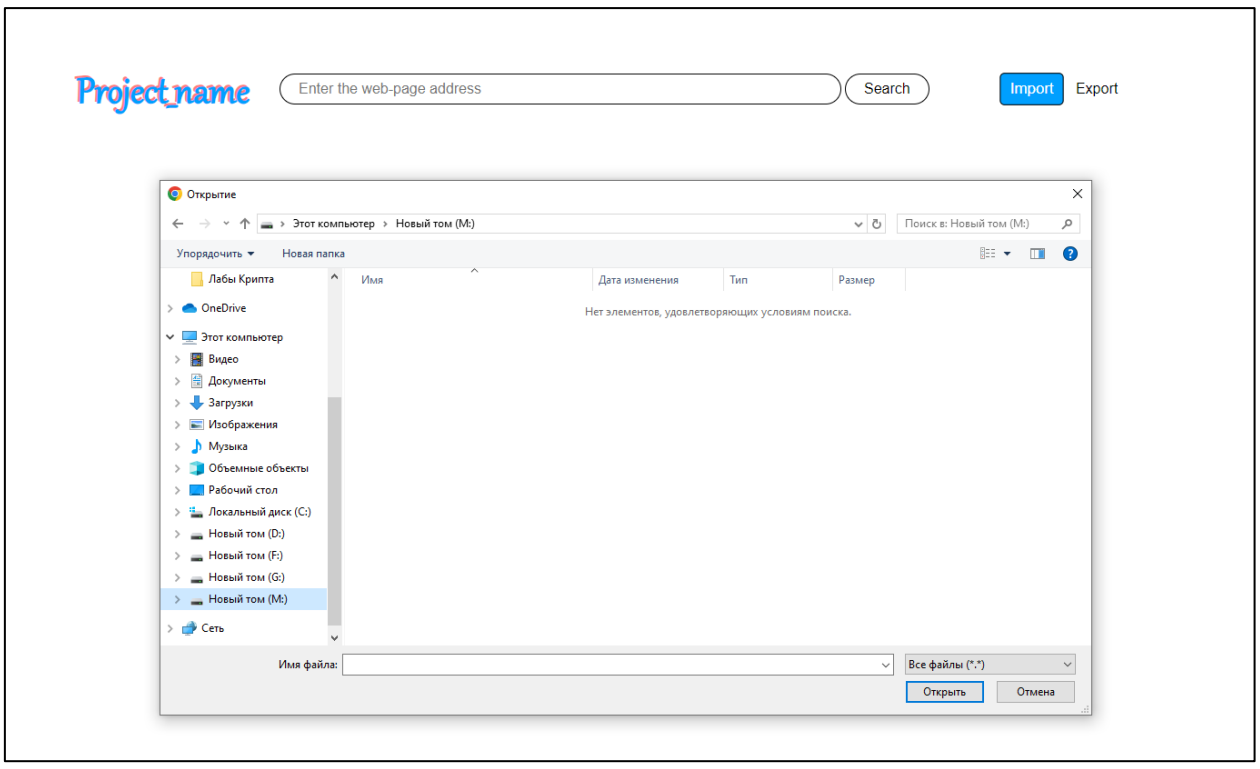


Рис.6 – Демонстрация импорта данных.



2.2.1. Импорт данных

2.2.1.1. Сценарий использования - “массовый импорт данных”

Действующее лицо - пользователь

Основной сценарий:

1. Пользователь нажимает на кнопку Import
2. Появляется окно выбора файла
3. Пользователь выбирает файл
4. Данные импортируются и выводятся в виде графа и таблицы

Альтернативный сценарий:

1. Пользователь не выбрал файл

2.2.1.2. Сценарий использования - “ручной импорт данных”

Действующее лицо - пользователь

Основной сценарий:

1. Пользователь вводит в строку поиска URL интересующей его странице.
2. Пользователь нажимает кнопку поиска.
3. Пользователь ожидает, пока сервер обработает запрос.
4. Данные выводятся в виде графа и таблицы.

Альтернативный сценарий:

1. Пользователь вводит неправильный URL
2. Данные по такому запросу не предоставляются

2.2.2. Представление данных

Действующее лицо - пользователь

Основной сценарий:

1. После успешного импорта данных пользователь переходит на вкладку “Граф”, где находится графическое представление данных.

Альтернативный сценарий :

1. После успешного импорта данных пользователь переходит на вкладку “Таблица”, где находится табличное представление данных.

2.2.3. Анализ данных

Действующее лицо - пользователь

Основной сценарий:

1. Пользователь переходит во вкладку “Advanced settings”
2. Пользователь выбирает одну из опций, представленных в списке
3. Если опция предполагает наличие дополнительных параметров, то пользователь выбирает их из списка в соответствующем появившемся окне.
4. После выбора опций, пользователь выбирает вкладку применить.
5. Пользователь переходит на вкладку с графическим представлением, чтобы зафиксировать результат.

2.2.4. Экспорт данных

Действующее лицо - пользователь

Основной сценарий:

1. Пользователь нажимает на кнопку “Export”
2. Данные из текущего состояния системы помещаются в файл фиксированного формата.
3. Файл скачивается на устройство пользователя.

2.3. Вывод об операциях

Преобладающей операцией будет являться чтение, так как количество операций записи не преобладает над количеством операций чтения. Основная функция, которую пользователь будет использовать является запрос графа по URL-ссылки web-страницы. Эта операция сопровождается как записью в базу

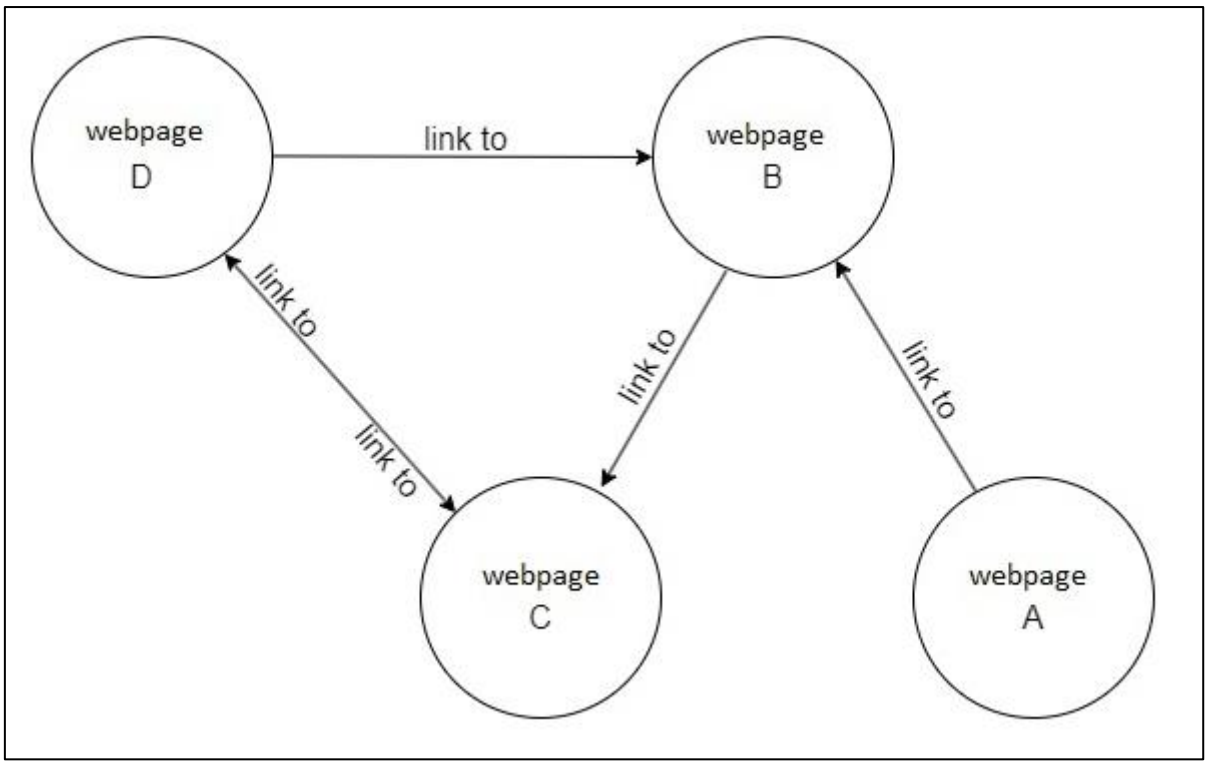
данных, так и чтением из нее. К тому же, пользователь может применить на возвращенный ему граф предоставленные алгоритмы, которые в большинстве случаев используют операции чтения из базы данных.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных - Neo4j

3.1.1. Графическое представление данных

Рис.7 – Графическое представление нереляционной базы данных



3.1.2. Описание назначений коллекций, типов данных и сущностей

1. Вершина (Узел) - сущность, соответствующая одной веб-страницы сайта

Хранимая о вершине информация:

- уникальный идентификатор (ID) - integer (4 байта)
- URL страницы - string (до 2048 байт)

- Дата и время обновления информации о вершине - DateTime (10 байт)
- Название веб-сайта, к которому веб-страница принадлежит (до 20 байт)

2. Ребро - сущность, содержащая информацию о том, какая веб-страница (откуда идет ребро) имеет гиперссылку на другую какую-то (куда идет ребро) веб-страницу:

Хранимая о ребре информация:

- уникальный идентификатор (ID) - integer (4 байта)
- Дата и время обновления информации о ребре - DateTime (10 байт)

3.1.3. Оценка удельного объема информации, хранимой в модели

Одна вершина занимает до ~2082 байт, ребро между двумя вершинами занимает 14 байтов.

Если задать ограничение для поиска "построение до x вершин", где каждая вершина будет иметь в среднем до 10 неповторяющихся ребер, то один такой поиск (при отсутствии кэша) создает x * (2082 + 10 * 14) = 2222x байт данных.

При наличии кэша часть данных или все данные о графе уже могут храниться в БД - в зависимости от уже имеющихся в нем данных. Пусть благодаря кэшу поиск будет создавать (2222x) * 9/10 = 1999,8x байт.

Пусть данные бд будут удаляться каждый месяц. Тогда если за сутки сервисом пользуются в среднем 10 человек, каждый из которых производит поиск в среднем 3-х веб-страниц, то в месяц серверу необходимо иметь 10 * 3 * 30 * 1999,8x = 1799820x свободных байт.

Тогда формула объема следующая: V(N) = 1799820 * N, где N - количество страниц, которые требуется вывести в рамках одного поиска. То есть, если N = 10000 страниц, то потребуется ~16,76 Гб, а если 30000 страниц ~50,29 Гб

3.1.4. Запросы к модели, с помощью которых реализуются сценарии использования

Вывести вершины (веб-страницы), относящиеся к заданному сайту

MATCH (n:< Website >) RETURN n

Поиск вершины

MATCH (n:< Website > {URL: "< url искомой вершины >"}) RETURN n

Поиск ребра

MATCH (n:< Website > {url: "< url1 >"})-[m:LEADS_TO]->(s:< Website >{url: "< url2 >"})

Добавление вершины

CREATE (site: < Website > {id: < id >, url: "< url >", datatime:"< datatime >"})

Добавление связи

MATCH (n:< Website >{url: "< url1 >"}) MATCH (s:< Website >{url: "< url2 >"}) CREATE (n)-[m:LEADS_TO]->(s) SET m.id = < id >, m.datatime = "< datatime >"

Очистка графа

MATCH (n:< Website >) DETACH DELETE n

Обновление вершины

MATCH (n:< Website >{url: " < url1 > "}) SET n.datatime = "< new_datatime >"

Обновление ребра

MATCH (n:< Website >{url: "< url1 >"})-[m:LEADS_TO]->(s:< Website >{url: "< url2 >"}) SET m.datatime = "< new_datatime >"

Показать весь граф

MATCH (n) RETURN (n)

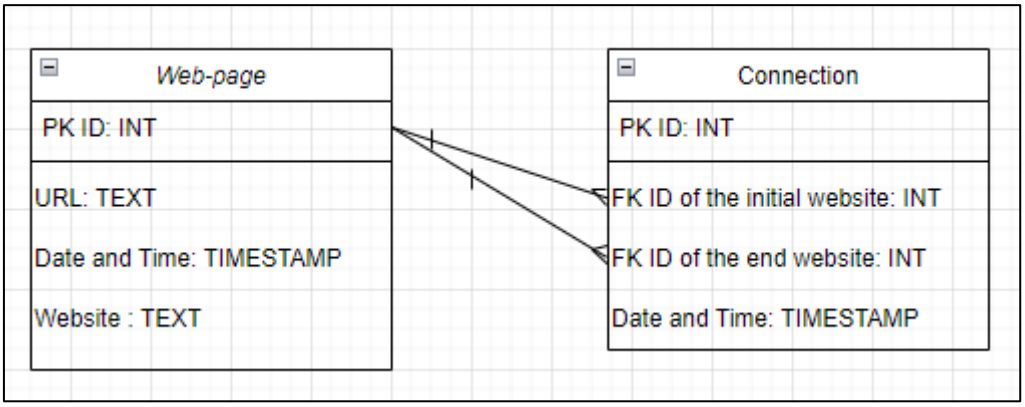
Показать часть графа, которая строилось в i-ый по счету поиска раз

MATCH (n:< Website >{id: i}) RETURN n

3.2. Реляционная модель данных

3.2.1. Графическое представление данных

Рис.8 – Графическое представление реляционной базы данных



3.2.2. Описание назначений коллекций, типов данных и сущностей

1. Таблица "Web-page"

- PK ID (уникальный идентификатор) - INT (4 байта)
- URL (страницы) - text (до 2048 байт)
- Date and Time (Дата и время обновления информации о странице) - timestamp (10 байт)
- Website - text (до 20 байт)

2. Таблица "Connection"

- PK ID (уникальный идентификатор) - INT (4 байта)
- FK ID of the initial website (ID сайта, откуда идет "связь") - INT (4 байта)
- FK ID of the end website (ID сайта, куда идет "связь") - INT (4 байта)
- Date and Time (Дата и время обновления информации о связи) - timestamp (10 байт)

3.2.3. Оценка удельного объема информации, хранимой в модели

Одна вершина занимает до ~2082 байт, ребро между двумя вершинами занимает 22 байта.

Если задать ограничение для поиска "построение до x вершин", где каждая вершина будет иметь в среднем до 10 ребер, то один такой поиск (при отсутствии кэша) создает x * (2082 + 10 * 22) = 2302x байт данных.

При наличии кэша часть данных или все данные о графе уже могут храниться в БД - в зависимости от уже имеющихся в нем данных. Пусть благодаря кэшу поиск будет создавать (2302x) * 9/10 = 2071,8 байт.

Пусть данные бд будут удаляться каждый месяц. Тогда если за сутки сервисом пользуются в среднем 10 человек, каждый из которых производит поиск в среднем 3-х веб-страниц, то в месяц серверу необходимо иметь 10 * 3 * 30 * 2071,8x = 1864620x свободных байт.

Тогда формула объема следующая: V(N) = 1864620 * N, где N - количество страниц, которые требуется вывести в рамках одного поиска. То есть, если x = 10000 вершин, то потребуется ~17,37 Гб, а если 30000 вершин ~52,1 Гб

3.2.4. Запросы к модели, с помощью которых реализуются сценарии использования

Вывести вершины (веб-страницы), относящиеся к заданному сайту (их принадлежность)

*SELECT * FROM Web_page WHERE website = '< website >';*

Поиск вершины

*SELECT * FROM Web_page WHERE URL = '< url искомой вершины >';*

Поиск ребра

*SELECT * FROM Connection WHERE ID_начальной_вершины = (SELECT ID FROM Web_page WHERE URL = '< url1 >') AND ID_конечной_вершины = (SELECT ID FROM Web_page WHERE URL = '< url2 >');*

Добавление вершины

INSERT INTO Web_page (ID, URL, Дата_и_время_обновления) VALUES (< id >, '< url >', '< datatime >');

Добавление связи

INSERT INTO CONNECTION (ID, ID_начальной_вершины, ID_конечной_вершины, Дата_и_время_обновления) VALUES (< id >, (SELECT ID FROM Web_page WHERE URL = '< url1 >'), (SELECT ID FROM Web_page WHERE URL = '< url2 >'), '< datatime >');

Очистка графа

DELETE FROM Connection; DELETE FROM Web_page;

Обновление вершины

UPDATE Web_page SET Дата_и_время_обновления = '< new_datatime >' WHERE URL = '< url1 >';

Обновление ребра

UPDATE Connection SET Дата_и_время_обновления = '< new_datatime >' WHERE ID_начальной_вершины = (SELECT ID FROM Web_page WHERE URL = '< url1 >') AND ID_конечной_вершины = (SELECT ID FROM Web_page WHERE URL = '< url2 >');

Показать весь граф

*SELECT * FROM Web_page; SELECT * FROM Connection;*

Показать часть графа, которая строилось в i-ый по счету поиска раз

*SELECT * FROM Web_page WHERE ID = i;*

3.3. Сравнение моделей

3.3.1. Удельный объем информации

Нереляционная модель требует меньшей памяти, чем реляционная: при N (ограничение на количество вершин) = 10000 вершин нереляционной необходимо ~16,76 Гб, а реляционной ~17,37 Гб, при N = 30000 вершин нереляционной - ~50,29 Гб, а реляционной ~52,1 Гб.

3.3.2. Запросы по отдельным юзкейсам

Реляционная модель требует в среднем больше запросов, чем нереляцонная, особенно для нахождения связей (откуда куда ведет) между сайтами.

3.3.3. Вывод

Нереляционная модель для данной задачи подходит лучше, чем реляционная (см. пункт "Сравнение моделей")

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1. Краткое описание

Клиентская часть разработана с использованием JavaScript. Для визуализации графа использовалась библиотеки force-graph, d3-force.

Серверная часть разрабатывалась на языке программирования python 3.10 с использованием flask.

В работе использовался Docker. Для базы данных и для web-приложения были созданы отдельные Docker-образы, из которых с помощью Docker-compose совместно запускаются соответствующие контейнеры, необходимые для работы приложения.

4.2. Использованные технологии

Neo4j - БД

Python 3.10 - Сервер

JavaScript, CSS – Клиентская часть

Docker

4.3. Снимки экрана приложения

Рис.9 – Графическое представление

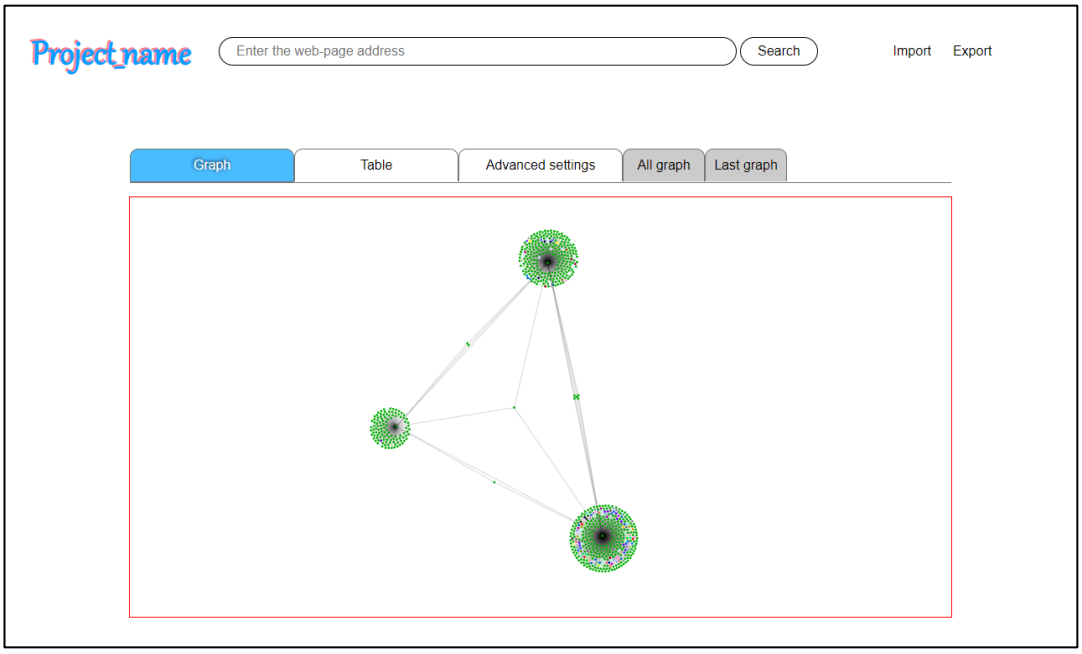


Рис.10 – Алгоритм поиска пути

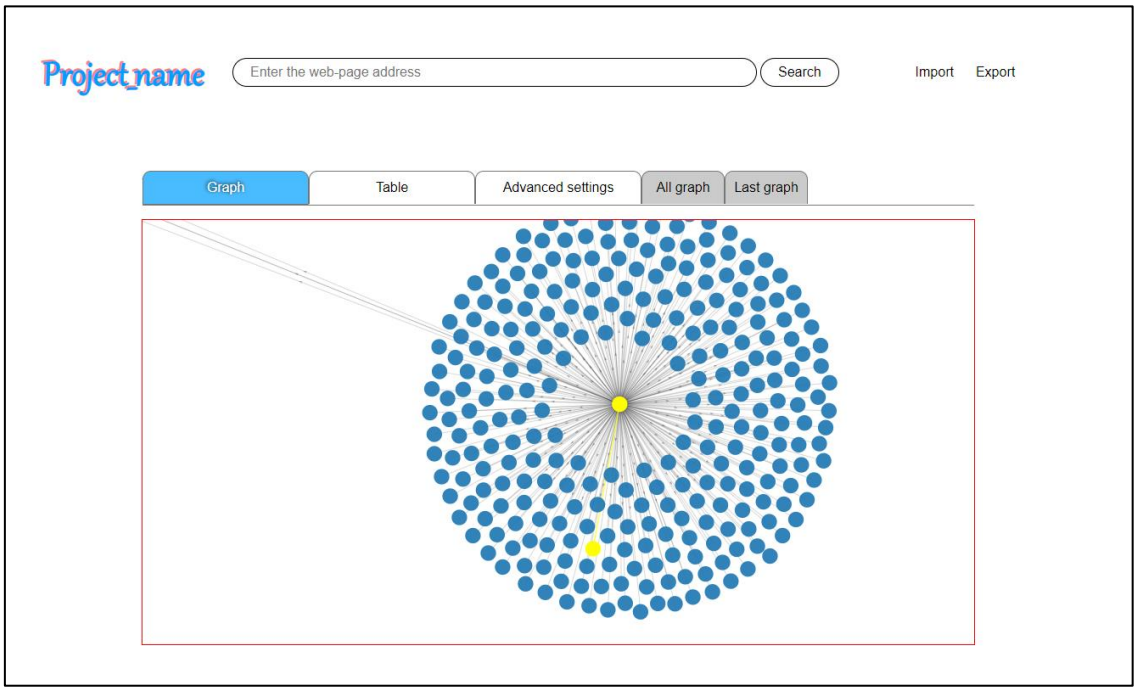
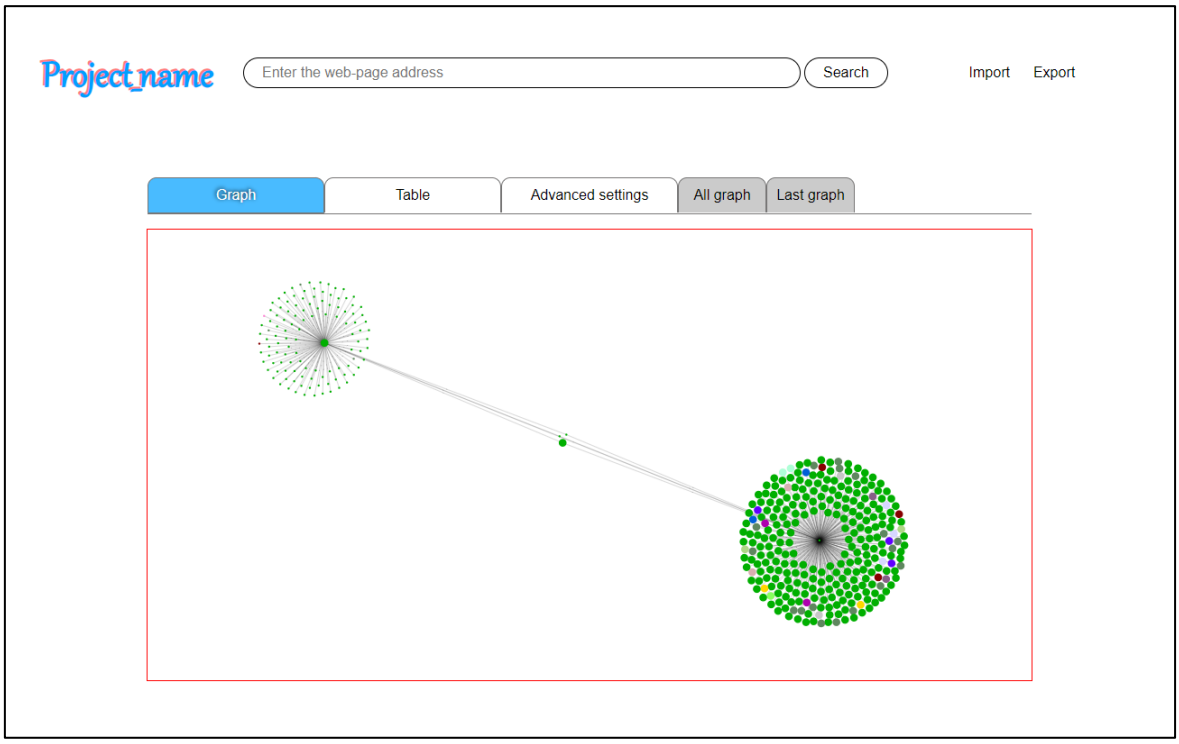
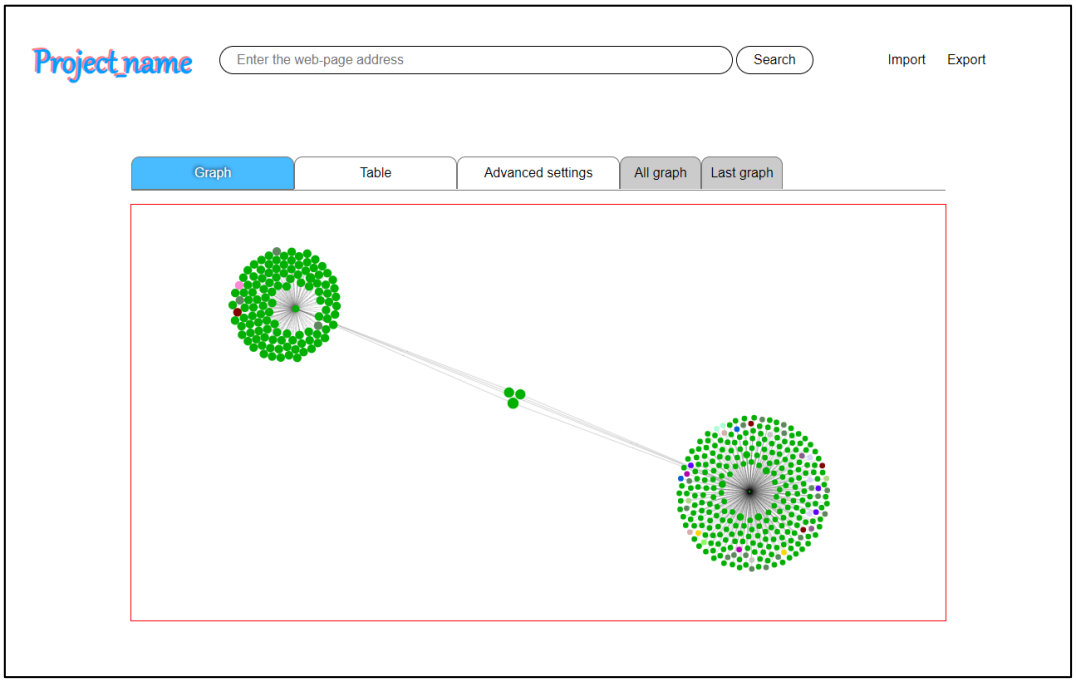


Рис.11 – Центральность



Puc.11 – Page Rank



5. ВЫВОДЫ

5.1. Достигнутые результаты

В ходе работы было разработано приложение предоставляющее возможность построения графа ссылок на web-страницах.с использованием нереляционной базы данных Neo4j. Для заданного пользователем адреса страницы строится граф и таблица ссылок. Полученный результат можно исследовать с помощью алгоритмов построения пути между вершинами, вычислением page rank’a и прочих опций.

5.2. Недостатки и пути для улучшения полученного решения

1) Приложение грузит ссылки медленно, скорость загрузки ссылок можно повысить.

2) Пользовательский интерфейс можно улучшить.

3) В текущей версии алгоритмы на графах не столь показательны в силу структуры построения (например, в большинстве случаев у одной вершины есть путь к 100+ потомком, из-за чего практически все пути построения сводятся к пути от этой вершины к потомкам). Можно изменить структуру построения графа.

5.3. Будущее развитие решения

В дальнейшем проект можно развивать, опираясь на недостатки, указанные в пункте 5.2.

6. ПРИЛОЖЕНИЯ

6.1. Документация по сборке и развертыванию приложения

Через Docker:

1) Сделать git clone проекта https://github.com/moevm/nosql2h23-web-graph/

2) Прописать docker-compose up и ожидать, пока решение соберется.

Без Docker:

1) Сделать git clone проекта https://github.com/moevm/nosql2h23-web-graph/

2) Должны быть установлены python 3.10, Nodejs (v16.17.1), pip

3) Сделать pip install -r requirements.txt

4) Установить neo4j версии 5.6.0 или 5.15 (версии должны быть enterprise)

5) Установить плагин APOC согласно инструкции из репозитория.

6) Установить плагин GDS: https://neo4j.com/graph-data-science-software/ Neo4j Graph Data Science Library 2.5.6 и перекинуть скаченный .jar в папку плагинов БД, в neo4j.conf добавить следующую строку: dbms.security.procedures.unrestricted=jwt.security.*,apoc.*,gds.*

7) Запустить из папки проекта final_version сервер командой python server.py

6.2. Инструкция для пользователя

1. Введите url интересующей вас странице в соответствующее поле и нажмите кнопку поиска.

2. Дождитесь, пока сервер обработает ваш запрос. После перейдите на одну из интересующих вас вкладок представления информации.

3. Для использования опций, перейдите в соответствующую вкладку и выберите одну из них. После чего нажмите кнопку применить и перейдите на вкладки представления информации.

4. Для импорта данных нажмите соответствующую кнопку в верхнем правом углу экрана. В появившемся окне выберите файл, который необходимо импортировать. Перейдите на вкладки представления информации.

5. Для экспорта данных нажмите соответствующую кнопку в верхнем правом углу экрана. В течении короткого времени начнется загрузка данных на ваше устройство.

7. ЛИТЕРАТУРА

1. GitHub - https://github.com/moevm/nosql2h23-web-graph/
2. Документация Neo4j - https://neo4j.com/docs/
3. Документация Docker - https://docs.docker.com/