**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: **Временная кластеризация исторических данных с привязкой к геокоординатам, выделение временных слоев (по векам/эпохам/периодам)**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 6303 |  | Ченцов Д.А. |
|  |  | Куликов М.Л. |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2019

**ЗАДАНИЕ**

|  |  |  |
| --- | --- | --- |
| Студенты: Ченцов Д.А., Куликов М.Л., группа 6303 | | |
| Тема проекта: временная кластеризация исторических данных с привязкой к геокоординатам, выделение временных слоев (по векам/эпохам/периодам). | | |
| Исходные данные: Объекты культурного наследия на территории Санкт-Петербурга.  Требуется реализовать приложение для кластеризации данных с использованием СУБД MongoDB. | | |
| Содержание пояснительной записки:   1. Содержание 2. Введение 3. Качественные требования к решению 4. Сценарии использования 5. Модель данных 6. Разработанное приложение 7. Выводы 8. Приложения 9. Литература | | |
| Предполагаемый объем пояснительной записки:  Не менее 25 страниц.  Дата выдачи задания: 15.02.2019 | | |
| Дата сдачи реферата: 3.06.2019  Дата защиты реферата: 3.06.2019 | | |
| Студенты гр. 6303 |  | Ченцов Д.А.  Куликов М.Л. |
| Преподаватель |  | Заславский М.М. |

**Аннотация**

В рамках данного курса требовалось разработать приложение с использованием нереляционной базы данных (или нескольких) на одну из поставленных тем. Была выбрана тема «Временная кластеризация исторических данных с привязкой к геокоординатам, выделение временных слоев (по векам/эпохам/периодам)».

**Summary**

As part of this course, it was necessary to develop an application using a non-relational database (or several) on one of the topics presented. The topic was chosen “  
Temporary clustering of historical data with reference to geo-coordinates”.

**содержание**

|  |  |  |
| --- | --- | --- |
| 1. | Введение | 5 |
| 2. | Качественные требования к работе | 6 |
| 3. | Сценарии использования | 6 |
| 4. | Модель данных | 10 |
| 5. | Разработанное приложение | 15 |
| 6. | Вывод | 16 |
| 7. | Приложения | 17 |
| 8. | Литература | 24 |

1. **введение**

Цель работы – создать приложение для кластеризации исторических данных, а также получения статистики по выделенным эпохам.

Было решено разработать WEB приложение на языке JS с использованием карт Google

Для организации хранения данных была выбрана СУБД MongoDB[1].

1. **КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ к работе**

Требуется разработать приложение с использованием СУБД MongoDB.

1. **СЦенарии использования**
2. ****
   1. **Макет интерфейса**

Рис. 1 – Макет интерфейса

* 1. **Описание сценариев использования**

## Сценарий использования - «Просмотр информации о достопримечательности на карте»

## Действующее лицо - Пользователь

* Основной сценарий:

1. Пользователь открывает приложение
2. Пользователь кликает на маркер
3. Пользователь получает информацию о достопримечательности

* Альтернативные сценарии:

1. Нет соединения с интернетом
2. Нет маркеров на карте

## Сценарий использования - «Применение фильтров поиска»

## Действующее лицо - Пользователь

* Основной сценарий:

1. Пользователь нажимает на кнопку Clustering
2. Из выпавшего меню пользователь выбирает нужные эпохи/эры, либо убирает их
3. Для установки глобального фильтра пользователь нажимает на кнопку settings и далее выбирает временные рамки. Для применения настроек, пользователь нажимает кнопку Accept
4. Пользователь получает уведомление о принятых настройках

## Сценарий использования - «Просмотр статистики по улицам»

## Действующее лицо - Пользователь

* Основной сценарий:

1. Пользователь нажимает на кнопку Clustering
2. В выпавшем меню пользователь нажимает на кнопку street statistics
3. Пользователь получает информацию о том, на каких трех улицах находится больше всего достопримечательностей

## Сценарий использования - «Импорт базы данных»

## Действующее лицо - Пользователь

* Основной сценарий:

1. Пользователь нажимает на кнопку Settings
2. Пользователь нажимает на кнопку Import
3. Пользователь получает уведомление о результате импорта

* Альтернативный сценарий:

1. Нет связи с сервером

## Сценарий использования - «Экспорт базы данных»

## Действующее лицо - Пользователь

* Основной сценарий :

1. Пользователь нажимает на кнопку Settings
2. Пользователь нажимает на кнопку Export
3. Пользователь получает уведомление о результате экспорта

* Альтернативный сценарий :

1. Нет связи с сервером

## Сценарий использования - «Удаление базы данных»

## Действующее лицо - Пользователь

* Основной сценарий :

1. Пользователь нажимает на кнопку Settings
2. Пользователь нажимает на кнопку Drop DB
3. Пользователь получает уведомление о результате сброса базы данных

* Альтернативный сценарий :

1. Нет связи с сервером

1. **МОдель данных**
   1. **Нереляционная модель**

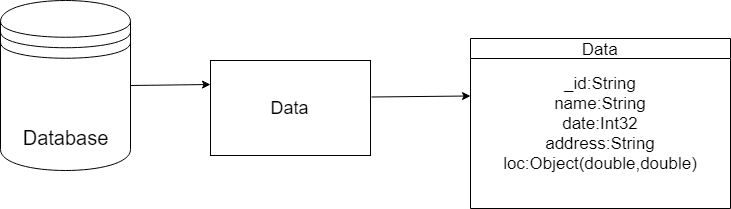


Рис. 2 – Нереляционная модель данных.

Данные хранятся в MongoDB. В базе данных всего одна коллекция data.

**Структура документа**

Для определения размера документа условимся, что у нас используются только ASCII-символы. Документ содержит:

1. Идентификатор
2. Название объекта
3. Год постройки
4. Адрес
5. Объект, содержащий координаты достопримечательности

{\_id:"...", name:"...", date:1234, address:"...", loc:{lat:0.000, lng:0.000}}

### Вычисление примерного объема данных

* id - Поле, которое автоматически генерируется MongoDB Vid = 4b
* Средняя длина поля name ~60 символов Vn = 120b
* Средняя длина поля address ~80 символов Va = 160b
* Поле date содержит числовое значение int32 Vd = 4b
* Поле loc представляет собой объект, имеющий 2 поля типа double Vl = 16b
* Подсчитаем средний размер документа Vdoc = 4+120+160+4+16 = 304b
* Подсчитаем размер всей коллекции N ~= 7700
* V = 304 \* 7700 = 2 340 800 ~ 2.2 mb
  1. **Запросы нереляционной модели**
* Запрос на добавление данных

|  |
| --- |
| dbo.collection("data").insertMany(data); |

Запрос 1.

* Запрос на поиск данных в периоде

|  |
| --- |
| db.data.find({date: {$gte: from, $lte : to}}); |

Запрос 2.

* Запрос на удаление данных

|  |
| --- |
| dbo.collection("data").drop({}); |

Запрос 3.

* 1. **Реляционная модель**

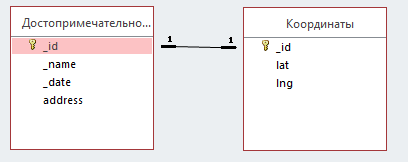


Рис. 3 – Реляционная модель данных.

Таблица "Достопримечательности":

* \_id – уникальный идентификатор
* \_name – Название достопримечательности
* address - Адрес
* Поле date – Год постройки

Таблица "Координаты":

* \_id – уникальный идентификатор.
* lat – широта
* lng - долгота
  1. **Оценка объема реляционной модели**

1. Таблица Достопримечательности:

* \_id - int = 4b
* \_name - string = 120b
* address - string = 160b
* Поле date - int32 = 4b

1. Таблица Координаты

* \_id - int = 4b
* lat - double = 8b
* lng - double = 8b
* Для одной записи понадобится Vdoc = 4+120+160+4+4+8+8 = 308b
* Для всех записей V = 7700\*Vdoc = 7700 \* 308 = 2 371 600b ~ 2.3 mb
  1. **Запросы реляционной модели**
* Запросы на добавление данных:

|  |
| --- |
| INSERT data(\_id, name, date, address) VALUES (\_id, name, date, address); INSERT coordinates(\_id, lat, lng) VALUES (\_id, lat, lng); |

Запрос 4.

* Запросы на поиск данных:

|  |
| --- |
| SELECT \* FROM Достопримечательности INNER JOIN Координаты ON Достопримечательности.[\_id] = Координаты.[\_id] WHERE Достопримечательности.date >= from AND Достопримечательности.date <= to; |

Запрос 5.

* 1. **Подсчет количества запросов**

N = 7700.

Среднее количество запросов при добавлении данных в nosql для одной записи = 5.

При добавлении данных в nosql необходимо 5N = 57700 = 38500 запросов.

Среднее количество запросов при добавлении данных в nosql для одной записи = 4(для первой таблицы) + 3(для второй таблицы) = 7 При добавлении данных в sql необходимо 7N = 77700 = 53900 запросов.

* 1. **Сравнение моделей**
* SQL модель данных требует больше места. Поскольку в SQL нет поддержки массивов, потребуется хранить гео-координаты в отдельных таблицах.
* В SQL модели требуется большее кол-во запросов для добавления записей (~ в 1.4 раза), по сравнению с NoSQL.

### Использование модели данных NoSQL более выгодно, т.к. при использовании NoSQL требуется меньше памяти для хранения и меньше запросов для поиска информации.

1. **Разработанное приложение**
   1. **Краткое описание**

Клиент-серверное веб приложение, в качестве сервера используется связка Express + MongoDB

* 1. **Использованные технологии**

**СУБД:** MongoDB

**Back-end:** NodeJS

**Front-end:** JS, JQuery, Google Maps API[3]

* 1. **Ссылки на приложение**

**GitHub: [2]**

1. **Вывод**
   1. **Достигнутые результаты**

В ходе работы было создано клиент-серверное приложение, позволяющее кластеризировать исторические данные.

* 1. **Недостатки и пути для улучшения полученного решения**

При большом количестве данных на карте происходят небольшие зависания, для исправления этого можно улучшить алгоритмы поиска данных и отрисовки на карте.

* 1. **Будущее развитие решения**

В будущем планируется добавить новые способы сбора и просмотра статистики (графики, диаграммы и т.п.)

1. **Приложения**
   1. **Документация по сборке и развертыванию**

## Необходимо проверить наличие:

1. MongoDB
2. NodeJS

## Последовательность запуска

1. Клонирование проекта
2. Установка необходимых пакетов командой npm install
3. Необходимо получить api key для карт google
4. Вставить свой ключ в строку подключаемого скрипта в файле /public/index.html вместо <YOUR\_KEY> (146 строка)
5. Запустить скрипт server.js
6. Открыть index.html в браузере
7. Открыть настройки и импортировать данные
   1. **Примеры работы программы**

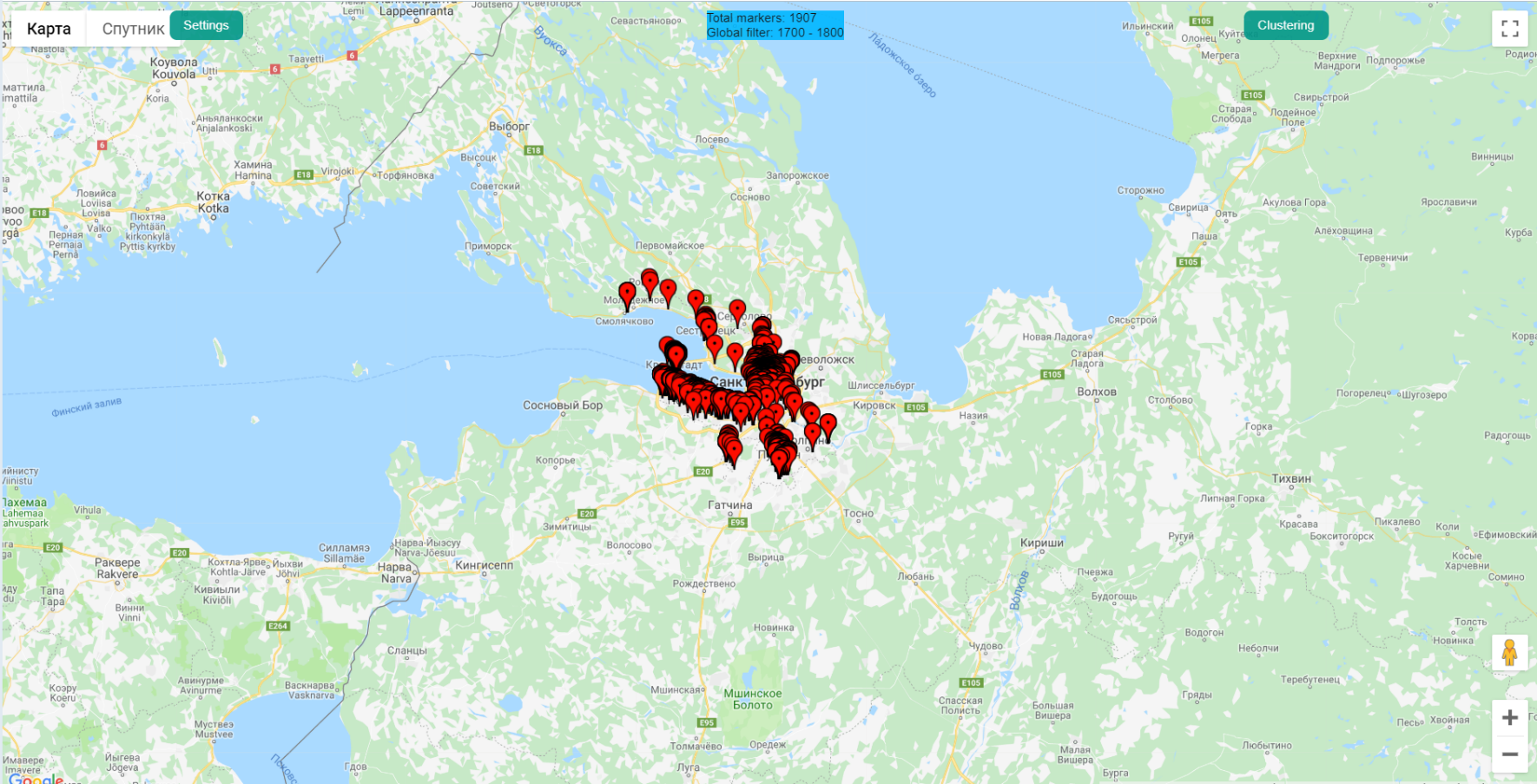


Рис. 5 – Главный экран

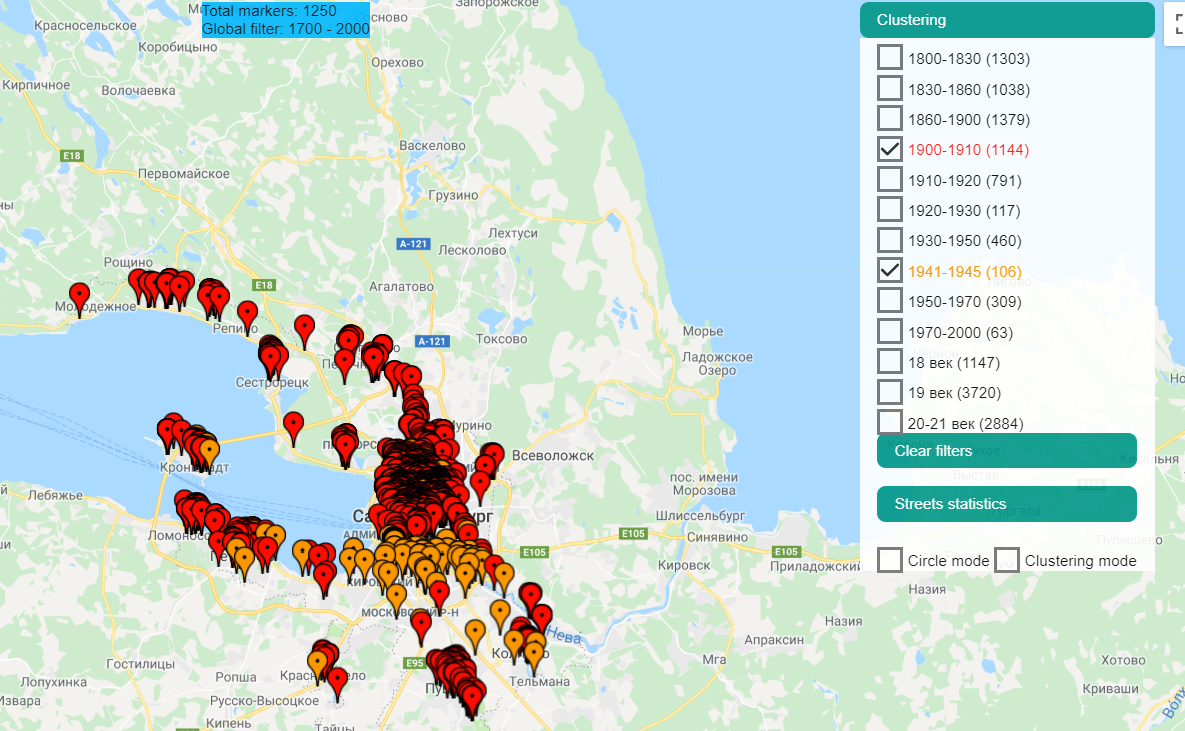


Рис. 6 – Главный экран с применением кластеризации по эпохам

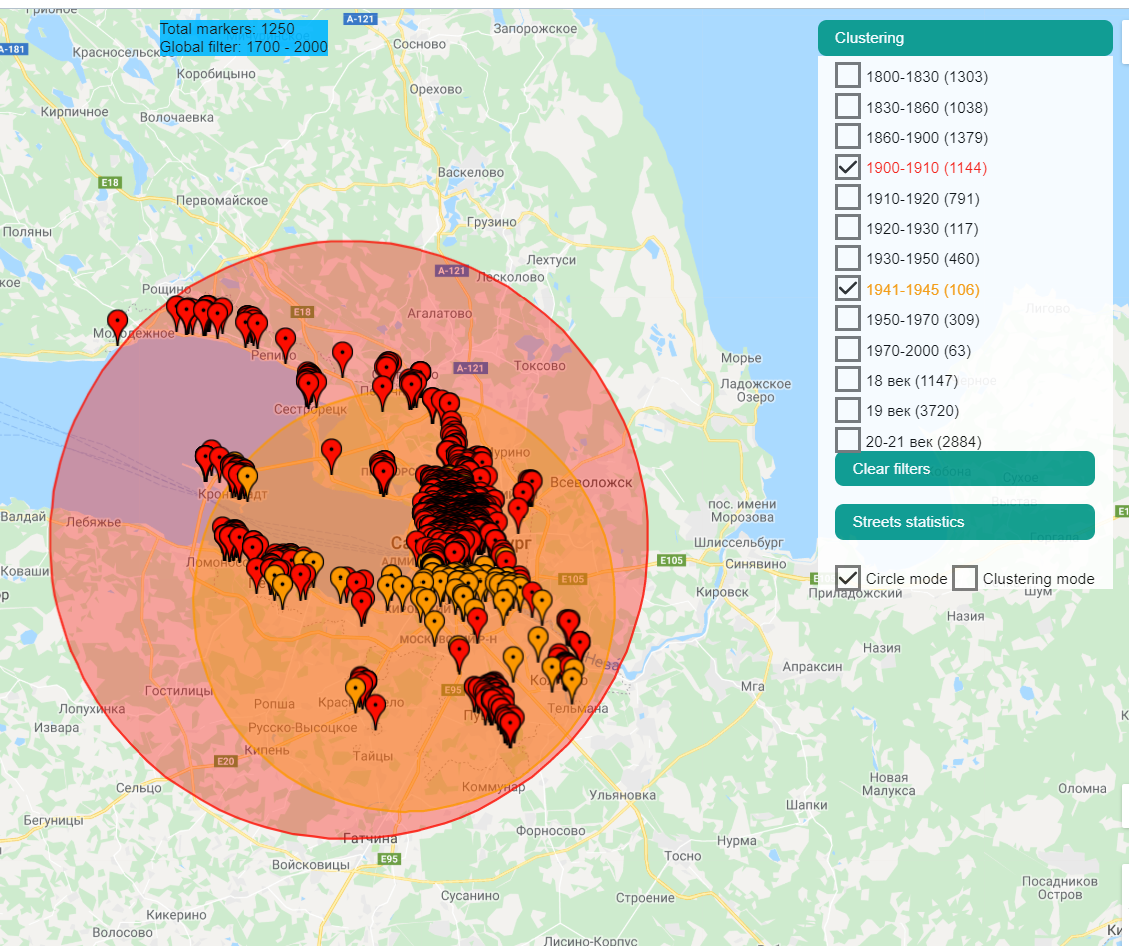


Рис. 7 – Зонирование

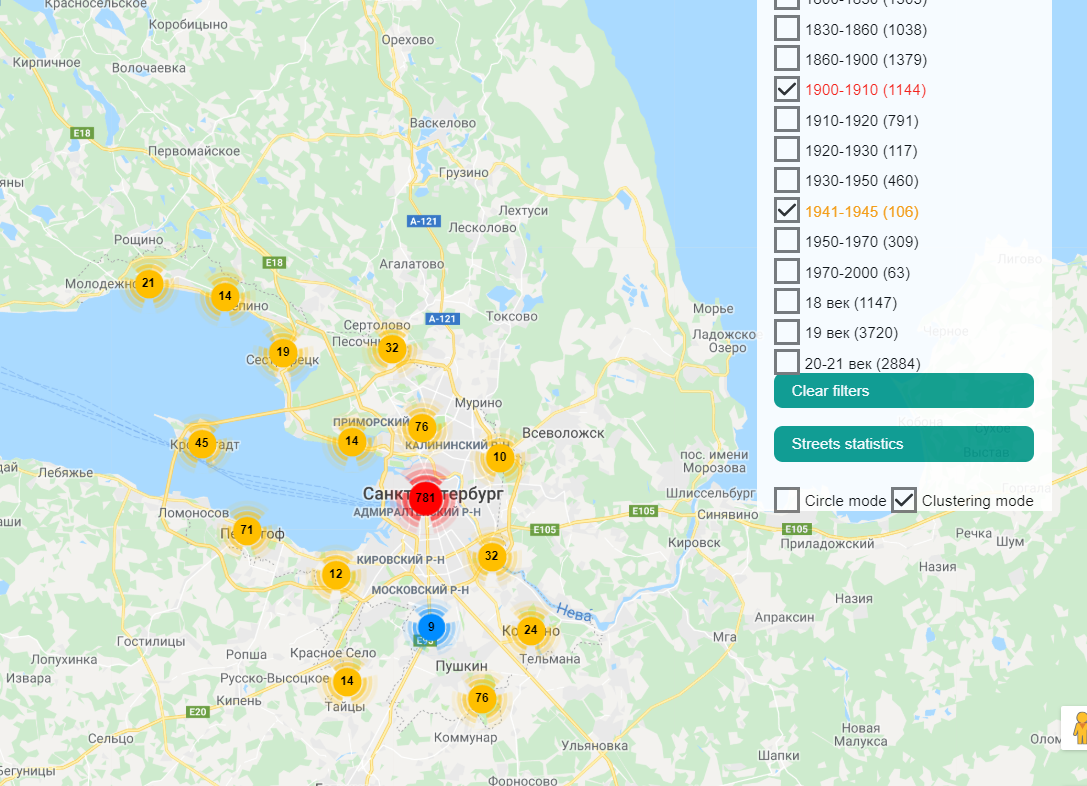


Рис. 8 – Режим кластеризации

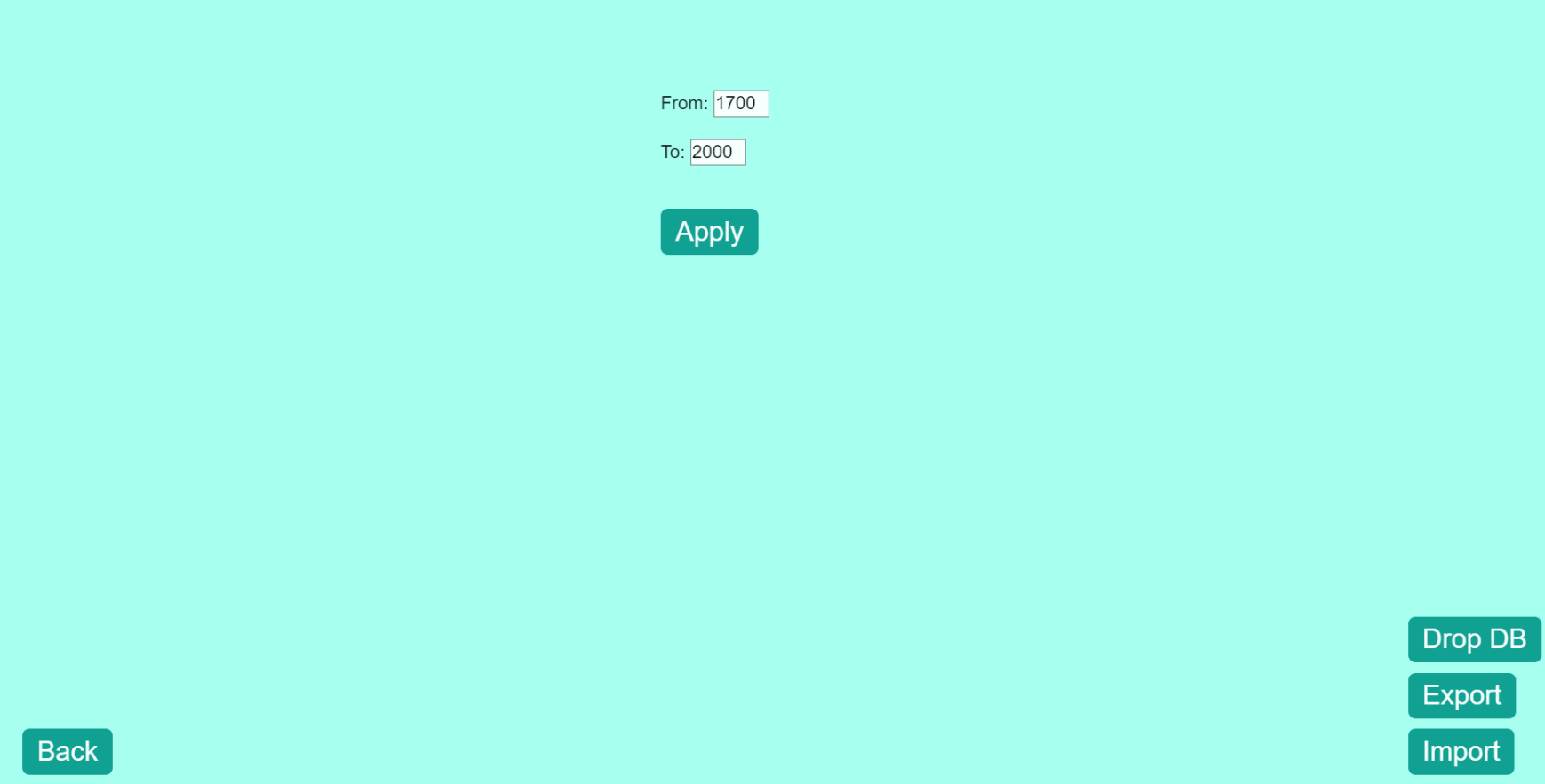


Рис. 9 – Страница настроек



Рис. 10 – Сброс данных, экспорт, импорт

1. **Литература**
2. The MongoDB Manual (дата обращения – 1.03.2019). URL: <https://docs.mongodb.com/manual/>
3. Исходный код проекта. URL: <https://github.com/moevm/nosql1h19-history-data>
4. Google Maps API URL: <https://cloud.google.com/maps-platform/maps/>