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 маршрутов без левых поворотов и разворотов

Студенты гр. 6303 _____ Иванов Д.В.

_____ Ильяшук Д.И.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2019

ЗАДАНИЕ

Студенты: Иванов Д.В., Ильяшук Д.И.

Группа 6303

Тема проекта: Построение маршрутов без левых поворотов и разворотов.

Исходные данные:

Требуется реализовать приложение для построения маршрутов без левых поворотов и разворотов 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: 29.05.2019

Дата защиты реферата: 29.05.2019

Студенты гр. 6303 _____ Иванов Д.В.
Ильяшук Д.И.

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 «Построение маршрутов без левых поворотов и разворотов».

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 “Building routes without left turns”.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Цель работы – создать приложение для построения маршрутов без левых поворотов и разворотов, а также сравнить их с традиционными маршрутами по таким параметрам как время и скорость, тем самым проверив теорию о том, что маршруты без левых поворотов зачастую могу оказаться быстрее, чем обычные.

Было решено разработать мобильное приложение для OS Android в качестве клиента и сервер в качестве back-end.

Для организации хранения данных была выбрана СУБД MongoDB.

2. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РАБОТЕ

Требуется разработать приложение с использованием СУБД MongoDB.

3. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

3.1. Макет интерфейса

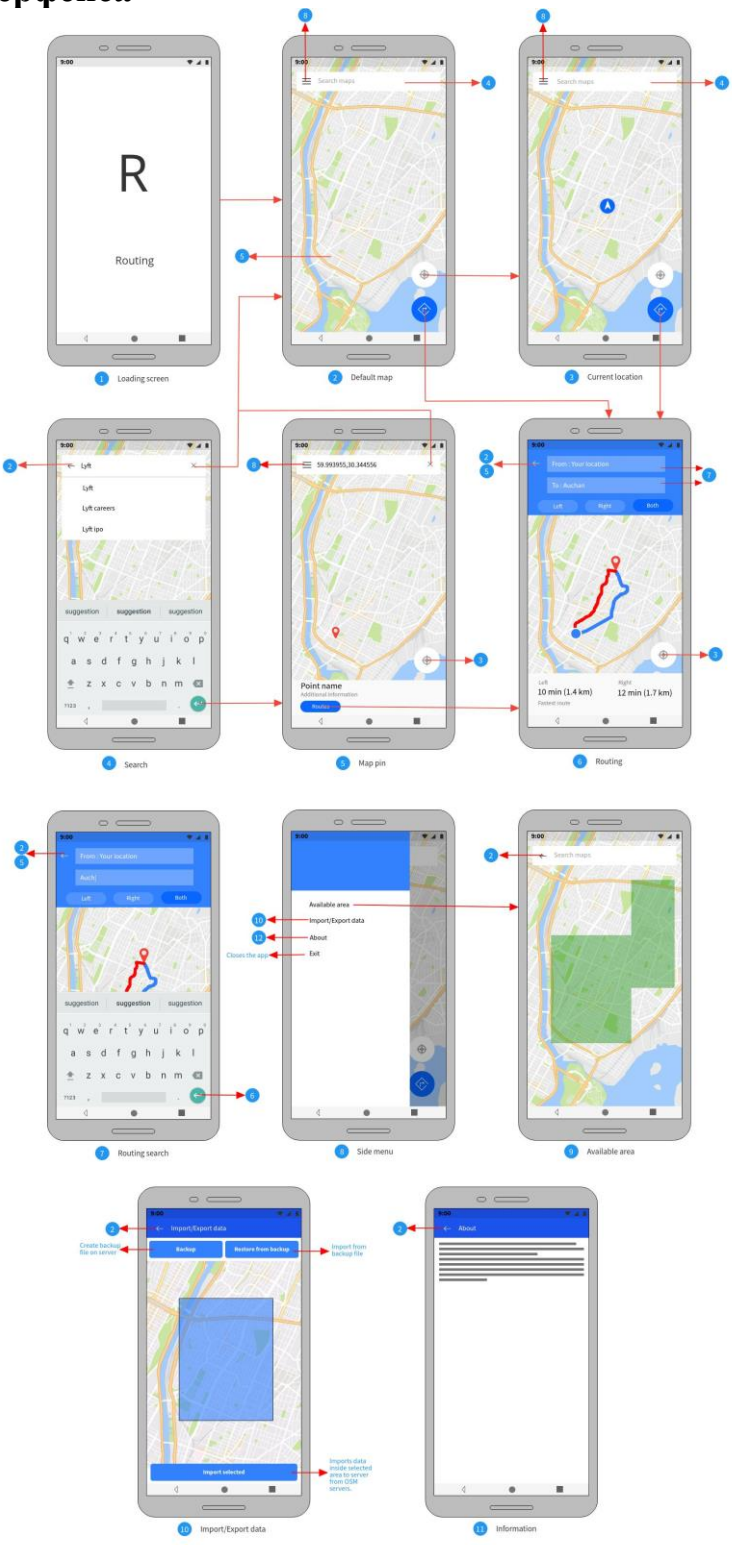


Рис. 1 – Макет интерфейса

3.2. Описание сценариев использования

3.2.1. Сценарий использования - «Поиск пути от местонахождения пользователя»

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит в приложение.
2. Пользователь указывает конечную точку с помощью поиска по карте либо долгим нажатием.
3. Пользователь наживает кнопку "Построить маршрут".
4. Пользователь сравнивает полученные маршруты и выбирает нужный ему.

Альтернативные сценарии:

1. Не удается определить местоположение пользователя.
2. Не удается найти точку с помощью поиска.
3. Не удается построить маршрут(ы).

3.2.2. Сценарий использования - «Поиск пути между двумя точками»

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит в приложение.
2. Пользователь указывает конечную точку с помощью поиска по карте либо долгим нажатием.
3. Пользователь наживает кнопку "Построить маршрут".
4. Пользователь указывает начальную точку с помощью поиска по карте.
5. Пользователь сравнивает полученные маршруты и выбирает нужный ему.

Альтернативные сценарии:

1. Пользователь наживает кнопку "Построить маршрут" без указания конечной точки.

3.2.3. Сценарий использования - «Определение местоположения пользователя»

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит в приложение.
2. Пользователь нажимает кнопку "Определить местоположение".
3. Пользователь получает точку со своим местоположением.

Альтернативный сценарий:

1. Не удается определить местоположение пользователя.

3.2.4. Сценарий использования - «Поиск по карте»

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит в приложение.
2. Пользователь активирует строку поиска.
3. Пользователь вводит поисковой запрос, состоящий из составляющих country, city, street, name, house member.
4. Пользователь выбирает в выпадающем меню нужную ему точку.

Альтернативный сценарий:

1. Пользователь не находит нужную ему точку.

3.2.5. Сценарий использования - «Получение доступной для построения маршрутов области»

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит в приложение.
2. Пользователь открывает боковое меню.
3. Пользователь выбирает пункт меню "Доступная область".
4. Пользователь получает доступную для построения маршрутов область, указанную зеленым цветом на карте.

3.2.6. Сценарий использования - «Импорт данных»

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит в приложение.
2. Пользователь открывает боковое меню.
3. Пользователь выбирает пункт меню "Импорт/Экспорт данных".
4. Пользователь выбирает область на карте и нажимает кнопку "Импорт выбранной области".
5. Через некоторое время выбранная область загружается на сервер.

Альтернативный сценарий:

1. Отсутствует соединение с сервером.

3.2.7. Сценарий использования - «Экспорт данных»

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит в приложение.
2. Пользователь открывает боковое меню.
3. Пользователь выбирает пункт меню "Импорт/Экспорт данных".
4. Пользователь нажимает кнопку "Создание резервной копии".
5. Резервная копия появляется на сервере.

Альтернативный сценарий:

1. Отсутствует соединение с сервером.

3.2.8. Сценарий использования - «Импорт данных из резервной копии»

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит в приложение.
2. Пользователь открывает боковое меню.
3. Пользователь выбирает пункт меню "Импорт/Экспорт данных".
4. Пользователь выбирает область на карте и нажимает кнопку "Восстановление данных из резервной копии".
5. Через некоторое время резервная копия восстановится на сервере.

Альтернативный сценарий:

1. Отсутствует соединение с сервером.

3.2.9. Сценарий использования - «Получение информации о приложении»

Действующее лицо: Пользователь

Основной сценарий:

1. Пользователь заходит в приложение.
2. Пользователь открывает боковое меню.
3. Пользователь выбирает пункт меню "О приложении".
4. Пользователь ознакомляется с полученной информацией.

3.2.10. Сценарий использования - «Выход из приложения»

Действующее лицо: Пользователь

Основной сценарий:

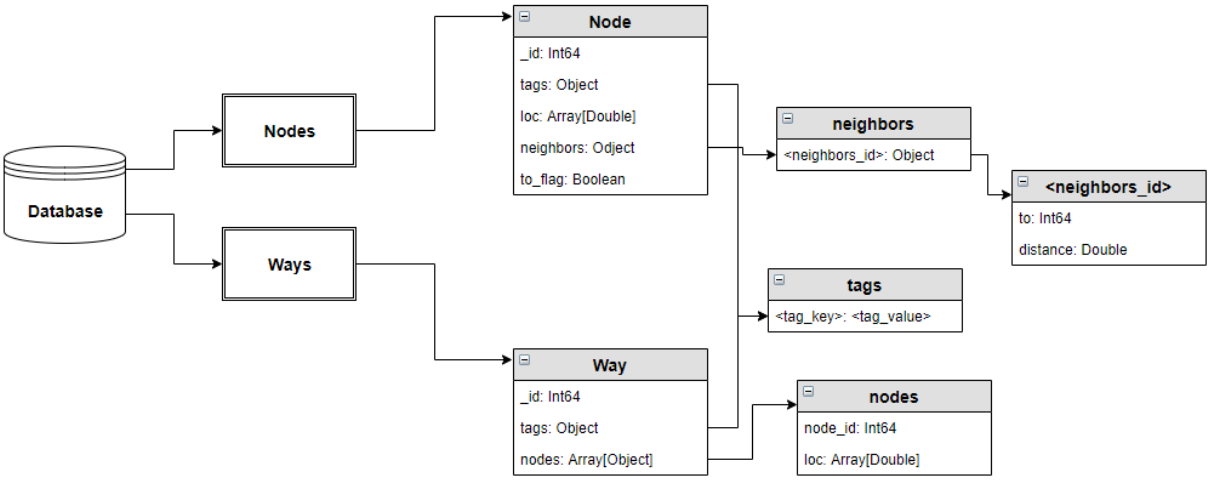
1. Пользователь уже находится в приложении.
2. Пользователь открывает боковое меню.
3. Пользователь выбирает пункт меню "Выход".
4. Пользователь подтверждает намерение выйти в открывшемся диалоговом окне.
5. Пользователь покинул приложение.

Альтернативный сценарий:

1. Пользователь не подтверждает намерение выйти в открывшемся диалоговом окне.

4. МОДЕЛЬ ДАННЫХ

4.1. Нереляционная модель



БД содержит две коллекции "nodes" и "ways"

- Коллекция "nodes"
  - _id - уникальный идентификатор узла
  - loc - координаты узла
  - tags - теги узла
    - key - ключ тега
      - value - значение тега
  - to_flag - флаг для обозначения всех найденных соседей
  - neighbors - информация об узлах-соседях
    - id узла-соседа
      - distance - расстояние до узла-соседа
      - way - дорога, соединяющая узлы

- Коллекция "ways"
  - _id - уникальный идентификатор пути
  - tags - теги пути
    - key - ключ тега
      - value - значение тега
  - nodes - информация об узлах, составляющих путь

- node_id - id узла
- loc - координаты узла

4.2. Оценка объема нереляционной модели

Коллекция "nodes":

- _id - тип Int64. **V = 8b**
- loc - тип [double, double]. **V = 16b**
- tags - тип Object. V = 2 * *Nv* * *Ntg* b, где *Nv* ~ *12* средняя длина значения тега, *Ntg* ~ 2, среднее количество тегов узла. **V = 48b**
- to_flag - тип Boolean. **V = 1b**
- neighbors - тип Object. V = (8 + 8) * *Nn*, где *Nn* ~ *2*, среднее количество соседей. **V = 32b**

Коллекция "ways":

- _id - тип Int64. **V = 8b**
- tags - тип Object. V = 2 * *Nv* * *Ntg* b, где *Nv* ~ *9* средняя длина значения тега, *Ntg* ~ *2*, среднее количество тегов узла. **V = 36b**
- nodes - тип Array(Object). V = (8 + 8 * 2) * *Nn*, где *Nn* ~ 7, среднее число узлов в пути. **V = 168b**

Средний объем узла **Vn = 105b**. Поскольку на практике лишь около 2% узлов имеют соседей (атрибут neighbors), и лишь около 5% имеют теги (атрибут tags), то Vn -> 24b. Средний объем пути **Vw = 212b**.

Объем данных для хранения Nn узлов и Nw путей:

- V(Nn, Nw) = Vn * Nn + Vw * Nw

4.3. Запросы нереляционной модели

- Запрос на добавление узлов:
  - `db.nodes.insert_many([{ '_id': id, 'tg': tags, 'ky': keys, 'loc': loc }])`

- Запрос на добавление путей:
  - `db.ways.insert_many([{ '_id': id, 'tg': tags, 'ky': keys, 'nd': nodes, 'loc': locs }])`
- Запрос на обновление данных об узле (добавление информации о соседях-узлах):
  - `db.nodes.update_one( {'_id': id_from}, { '$push': { 'to': id_to, 'distances': length, 'ways': way_id } } )`
- Запрос на обновление данных об узле (поле 'to_flag' для обозначения всех найденных узлов-соседей):
  - `db.nodes.update( {'_id': node}, { '$set': {'to_flag': True} } )`
- Запрос для поиска узла по его id:
  - `db.nodes.find_one({'_id': node})`
- Запрос для поиска узла с id1, имеющего узел-сосед с id2:
  - `db.nodes.find_one({ '_id': id1, 'to': id2 })`
- Запрос для поиска пути, имеющего определенный тег и узел:
  - `db.ways.find({ 'tg': {'$in': tags}, 'nd': node_id })`
- Запрос для подсчета кол-ва путей с id не равным определенному, имеющих определенный тег и узел:
  - `db.ways.count_documents({ '_id': {'$ne': id}, 'nd': node_id, 'tg': {'$in': tags} })`

4.4. Реляционная модель

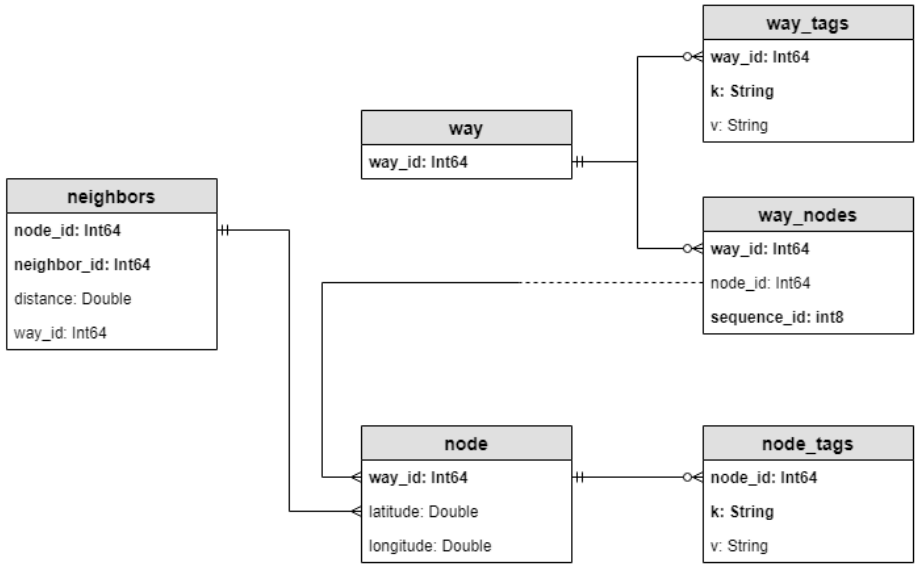


Таблица "nodes":

- **node_id** - уникальный идентификатор узла.
- latitude - широта координаты узла.
- longitude - долгота координаты узла.
- to_flag - флаг для обозначения всех найденных соседей.

Таблица "nodes_tags":

- **node_id** - уникальный идентификатор узла.
- **k** - ключ тега узла.
- v - значение тега узла.

Таблица "ways":

- **way_id** - уникальный идентификатор пути.

Таблица "ways_tags":

- **way_id** - уникальный идентификатор пути.
- **k** - ключ тега пути.
- v - значение тега пути.

Таблица "way_nodes":

- **way_id** - уникальный идентификатор пути.
- **sequence_id** - порядковый номер узла в пути.
- node_id - id узла.

Таблица "neighbors":

- **node_id** - уникальный идентификатор узла.
- **neighbor_id** - id узла-соседа.
- distance - расстояние между узлами.
- way_id - id пути, соединующая узлы.

4.5. Оценка объема реляционной модели

Таблица "nodes":

- **node_id** - тип Int64. **V = 8b**
- latitude - тип Double. **V = 8b**
- longitude - тип Double. **V = 8b**
- to_flag - **V = 1b**

Таблица "nodes_tags":

- **node_id** - тип Int64. **V = 8b**
- **k** - тип String. V = 2 * *Nnk*, где *Nnk* ~ 7, средняя длина ключа тега. **V = 14b**
- v – тип String. V = 2 * *Nnv*, где *Nnv* ~ 12, средняя длина значения тега. **V = 24b**

Таблица "ways":

- **way_id** - тип Int64. **V = 8b**

Таблица "ways_tags":

- **way_id** - тип Int64. **V = 8b**
- **k** - тип String. V = 2 * *Nwk*, где *Nwk* ~ 9, средняя длина ключа тега. **V = 18b**
- v - тип String. V = 2 * *Nwv*, где *Nwv* ~ 9, средняя длина значения тега. **V = 18b**

Таблица "way_nodes":

- **way_id** - тип Int64. **V = 8b**
- **sequence_id** - тип Int32. **V = 4b**
- node_id - тип Int64. **V = 8b**

Таблица "neighbors":

- **node_id** - уникальный идентификатор узла. Тип Int64. **V = 8b**
- **neighbor_id** - тип Int64. **V = 8b**
- distance - тип Double. **V = 8b**
- way_id - тип Int64. **V = 8b**

Для одного узла потребуется, в среднем:

- одна запись в таблице "nodes" = 25b
- две записи в таблице "node_tags" = 92b
- две записи в таблице "neighbors" = 64b

Для одного пути потребуется, в среднем:

- одна запись в таблице "ways" = 8b
- две записи в таблице "way_tags" = 88b
- семь записей в таблице "way_nodes" = 140b

Средний объём узла **Vn = 180b**, поскольку на практике лишь около 2% узлов имеют соседей (атрибуты to, distances, ways), и лишь около 5% имеют теги (атрибуты tg, ky), то Vn -> 25b. Средний объем пути **Vw = 236b**.

Объем данных для хранения Nn узлов и Nw путей:

- V(Nn, Nw) = Vn * Nn + Vw * Nw

4.6. Запросы реляционной модели

- Запросы на добавление узлов:
  - ```
    INSERT nodes(node_id, latitude, longitude)
    VALUES (node_id1, latitude1, longitude1), (node_id2, latitude2,
    longitude2);
    ```
  - ```
    INSERT nodes_tags(node_id, k, v)
    VALUES (node_id1, k1, v1), (node_id2, k2, v2);
    ```
- Запросы на добавление путей:
  - ```
    INSERT ways(way_id)
    VALUES (way_id1), (way_id2);
    ```
  - ```
    INSERT way_tags(way_id, k, v)
    VALUES (way_id1, k1, v1), (way_id2, k2, v2);
    ```
  - ```
    INSERT way_nodes(way_id, sequence_id, node_id)
    VALUES (way_id1, sequence_id11, node_id11), (way_id1,
    sequence_id12, node_id12), (way_id2, sequence_id21, node_id21),
    (way_id2, sequence_id22, node_id22);
    ```
- Запрос на добавление данных о соседе узла:
  - ```
    INSERT neighbors(node_id, neighbor_id, distance, way_id)
    VALUES (node_id1, neighbors_id1, distance1, way);
    ```

- Запрос на обновление данных об узле (поле 'to_flag' для обозначения всех найденных узлов-соседей):
  - `UPDATE NODES SET to_flag = true WHERE node_id = id;`
- Запрос для поиска узла по его id:
  - `SELECT * FROM NODES WHERE node_id = id;`
- Запрос для поиска узла с id1, имеющего узел-сосед с id2:
  - `SELECT * FROM NODES WHERE node_id = id1 AND neighbor_id = id2;`
- Запрос для поиска пути, имеющего определенный тег и узел:
  - ```
    SELECT * FROM WAYS
    LEFT JOIN way_tags ON way_id = way_tags.way_id
    WHERE way_id = id AND way_tag.v IN ("tag1", "tag2");
    ```
- Запрос для подсчета кол-ва путей с id не равным определенному, имеющих определенный тег и узел.
  - ```
    SELECT COUNT(*) FROM WAYS
    JOIN way_tags ON way_id = way_tags.way_id
    JOIN way_nodes ON way_id = way_nodes.way_id
    WHERE way_id != id AND v IN ("tag1", "tag2") AND node_id = n_id;
    ```

Кол-во необходимых запросов для поиска пути в SQL:

- Глубина 1 ~ 23 запроса
  - 1 запрос на проверку наличия всех соседей.
  - 1 запрос на поиск всех инцидентных узлу дорог.
  - *Nn * Nnb* запросов для поиска соседних узлов, где *Nn ~ 5*, среднее кол-во промежуточных узлов между соседними узлами, *Nnb ~ 3*, среднее кол-во соседних узлов.
  - *2 * Nnb* запросов на запись соседнего узла, где *Nnb ~ 3*, среднее кол-во соседних узлов.
- Глубина 2 ~ 92 запроса
  - *N1* запросов для начального узла, где *N1 = 23*, кол-во запросов глубины 1.
  - *N1 * Nnb* запросов для соседних узлов начального узла, где *N1 = 23*, кол-во запросов глубины 1, *Nnb ~ 3*, среднее кол-во соседних узлов.
- Глубина n = *N(n-1) + N1 * Nnb^(n-1)* запросов.

- *N(n-1)* запросов для узлов с глубиной n-1, где *N(n-1)* кол-во запросов глубины n-1.
- *N1 * Nnb^(n-1)* запросов для соседей узлов глубины n-1, где *N1 = 23*, кол-во запросов глубины 1, *Nnb ~ 3*, среднее кол-во соседних узлов одного узла.

4.7. Сравнение моделей

- SQL модель данных требует больше места. Поскольку в SQL нет поддержки массивов, потребуется хранить теги узлов, путей и составляющие узлы в отдельных таблицах, что добавляет значительные накладные расходы за счёт дублирования информации.
- В SQL модели требуется большее кол-во запросов для добавления записей об узлах и путях (в 2-3 раза), по сравнению с MongoDB.

Для добавления карты среднего объема (150000 nodes, 50000 ways) в БД потребуется:

- SQL - 450 запросов.
- noSQL - 200 запросов.

Для поиска маршрута средней длины (50 nodes) потребуется:

- SQL ~ 5000 запросов.
- noSQL ~ 2000 запросов.

Из приведенных рассуждений можно сделать вывод, что noSQL модель лучше, поскольку SQL имеет большое дублирование информации, больший расход места и зачастую требует большее количество запросов.

5. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

5.1. Краткое описание

Клиент-серверное приложение, в качестве сервера используется связка Flask + Python + MongoDB, клиент – Android-приложение.

5.2. Схема экранов приложения

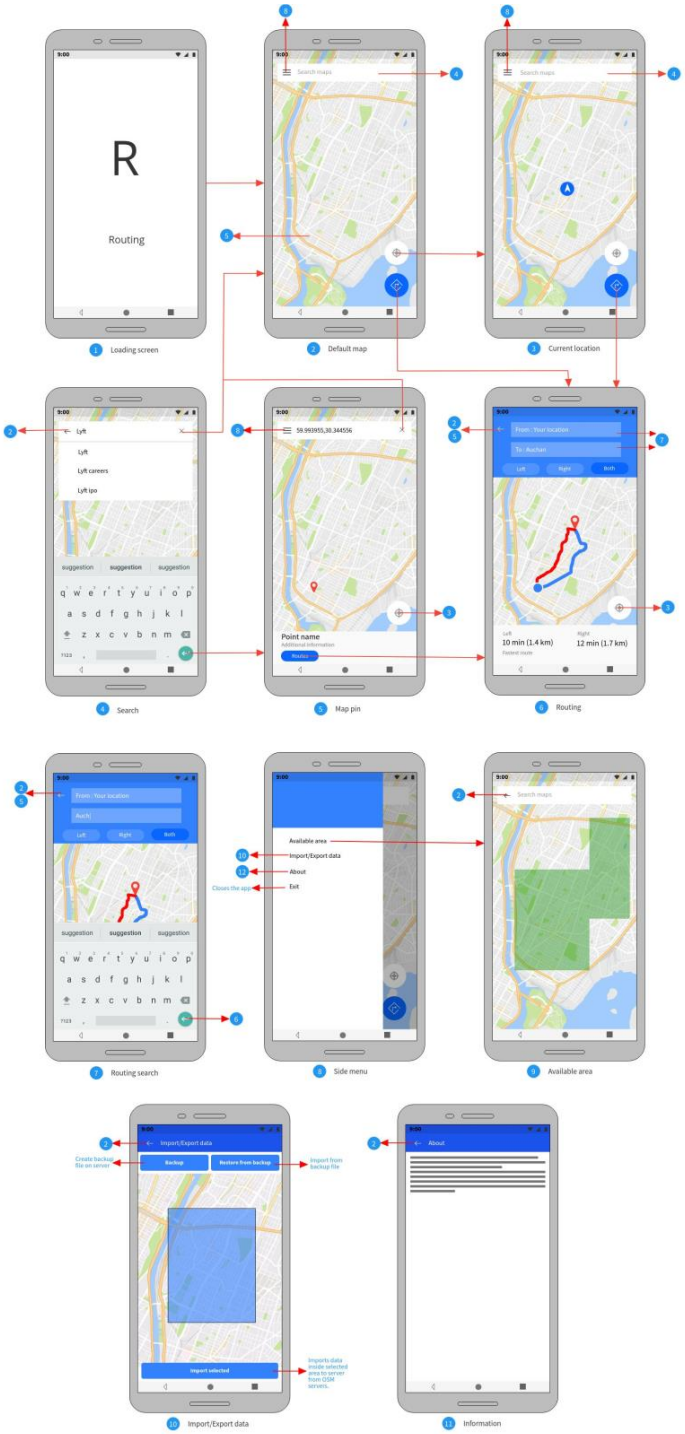


Рис. 2 – Схема экранов приложения

5.3. Использованные технологии

СУБД: MongoDB

Back-end: Python 3.7, Flask

Front-end: Android, Kotlin

5.4. Ссылки на приложение

GitHub: https://github.com/moevm/nosql1h19-right-route/wiki

6. ВЫВОД

6.1. Достигнутые результаты

В ходе работы было создано клиент-серверное приложение, строящее маршруты без левых поворотов и разворотов, с использованием данных OpenStreetMaps, хранимых в MongoDB.

6.2. Недостатки и пути для улучшения полученного решения

Использование документно-ориентированных СУБД вроде MongoDB для решения задач маршрутизации (работы с графами) является не самым лучшим вариантом, поэтому для большей результативности и простоты рациональнее использовать графовые СУБД (например, Neo4j).

6.3. Будущее развитие решения

Реализованная серверная часть приложения позволяет разработать веб-приложение.

7. ПРИЛОЖЕНИЯ

7.1. Документация по сборке и развертыванию

0) Скачать проект из репозитория (указан в ссылках на приложение)
1) Развертывание частей приложения:
   - Серверная часть:
     - Создать файл *config.json* из файла *settings/config_example.json*, заполнив поля “IP” и “port”.
     - Запустить MongoDB (используя стандартные настройки).
     - Установить необходимые python-зависимости из *requirements.txt*
     - Запустить файл *server.py* из каталога *source*, используя команду *python3.7 server.py*.
   - Android-приложение
     - Выполнить команду *gradlew assembleDebug*
     - Apk-файл может быть найден в *app/build/outputs/apk/app-debug.apk*

7.2. Снимки экрана приложения

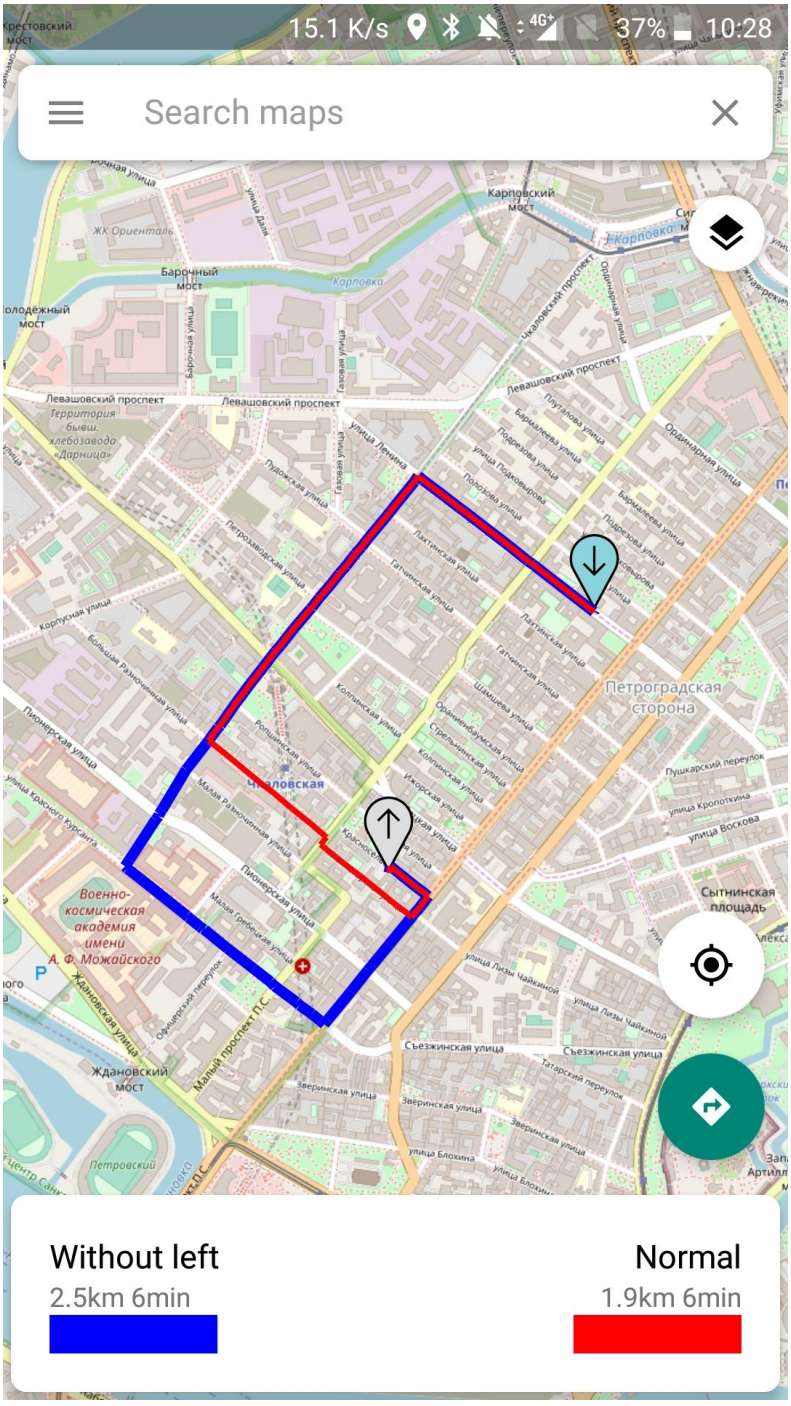


Рис. 3 – Главный экран с построенными маршрутами

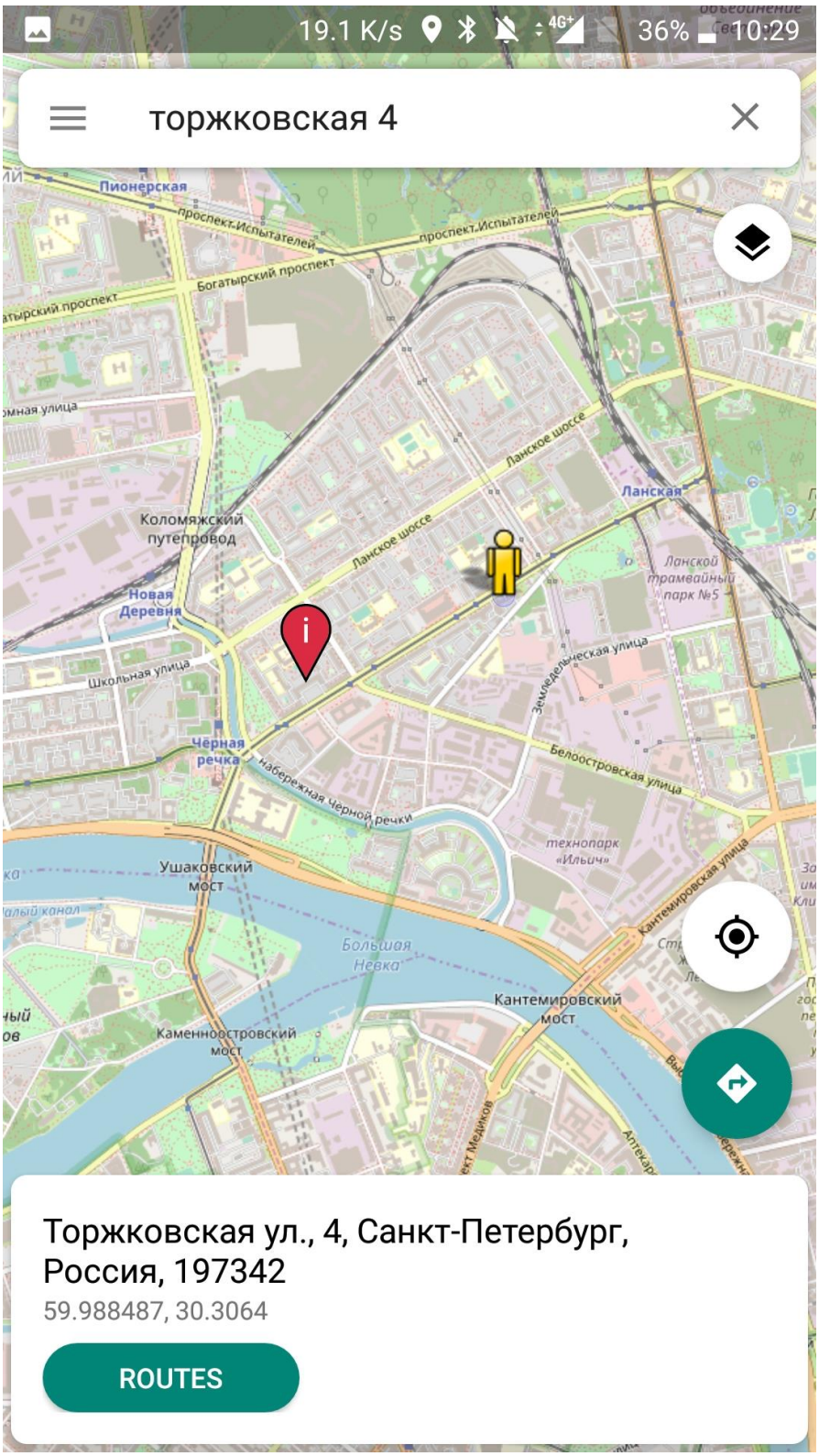


Рис. 4 – Главный экран с найденным местом

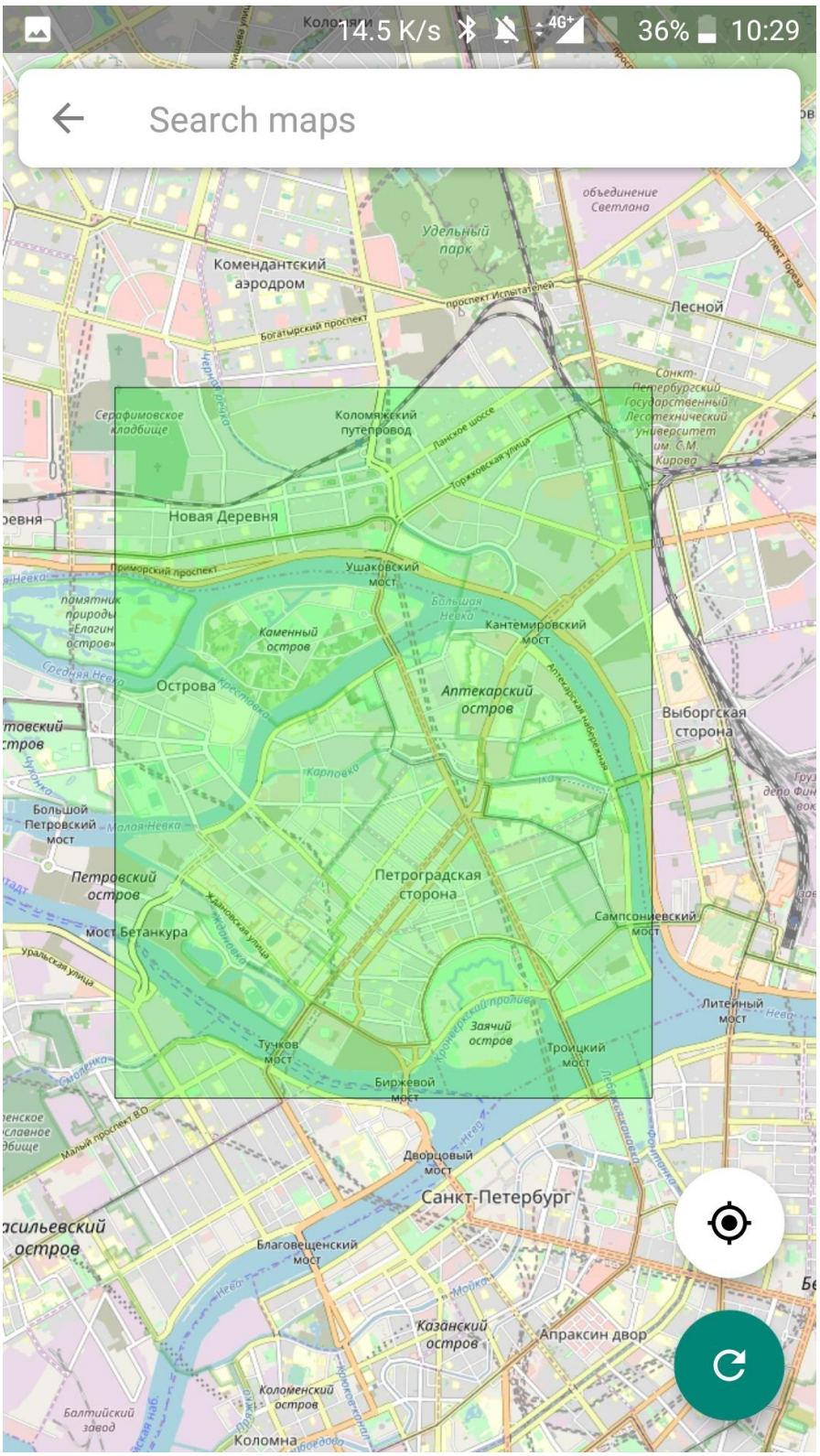


Рис. 5 – Доступная область

Рис. 6 – Импорт/экспорт данных

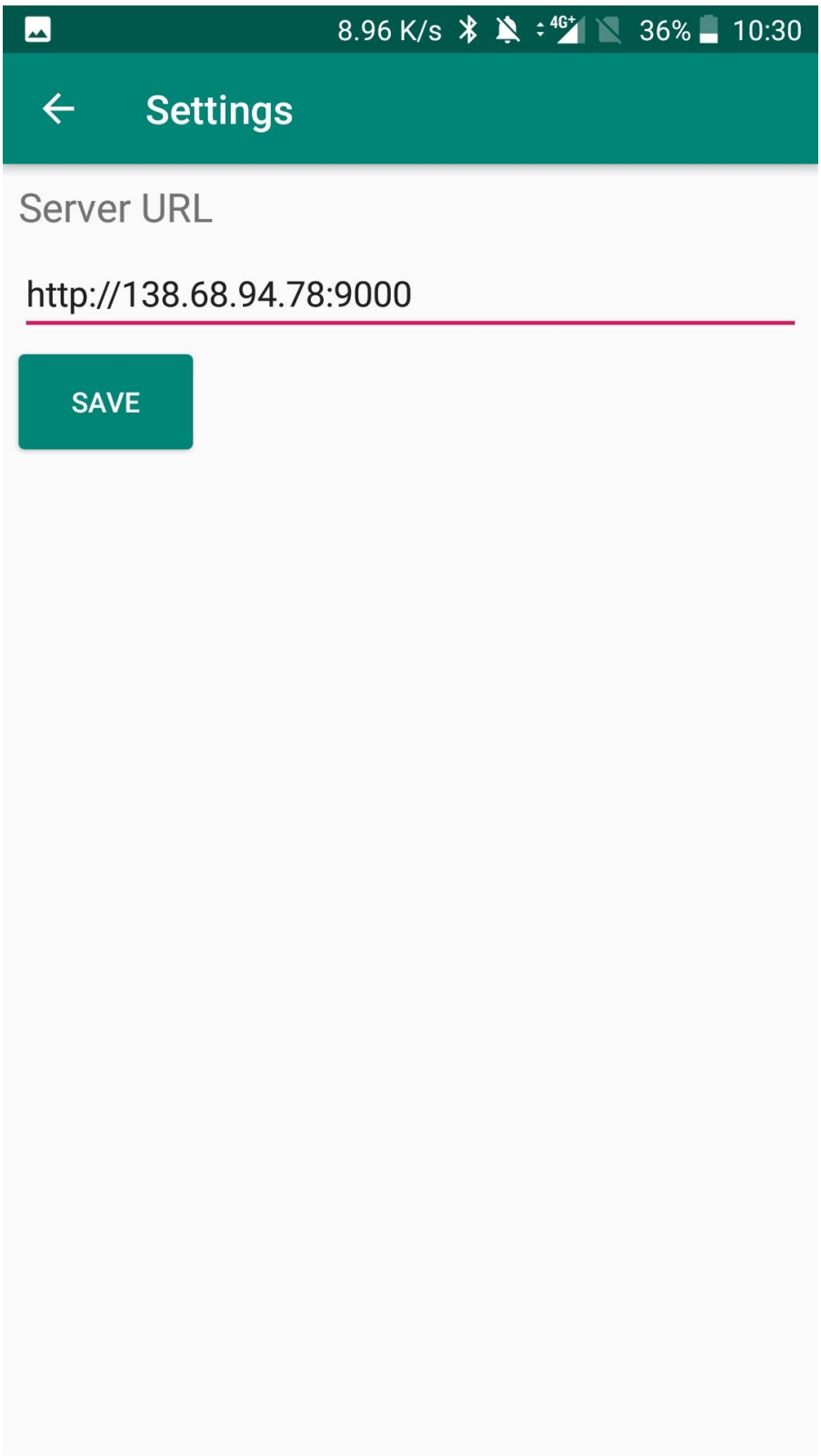


Рис. 7 – Страница настроек

8. ЛИТЕРАТУРА

1. Android developers (дата обращения – 07.03.2019). URL: https://developer.android.com/
2. The MongoDB Manual (дата обращения – 25.02.2019). URL: https://docs.mongodb.com/manual/