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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Разработка ПО информационных систем»

Тема: Построение маршрутов по рекам и каналам

Студенты гр. 0381 _____ Странникова Н.
_____ Ефимов Н.
_____ Табаков П.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2023

ЗАДАНИЕ

Студенты
Странникова Н.
Ефимов Н.
Табаков П.
Группа 0381

Тема проекта: Построение маршрутов по рекам и каналам.

Исходные данные:
Необходимо реализовать приложение для построение маршрутов для водной экскурсии с использованием графовой базы данных Neo4j с возможностью хранить уже построенные маршруты.

Содержание пояснительной записки:
«Содержание»
«Введение»
«Качественные требования к решению»
«Сценарий использования»
«Модель данных»
«Разработка приложения»
«Вывод»
«Приложение»

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: сентябрь 2023 г.
Дата сдачи реферата: 23.12.23
Дата защиты реферата: -

Студенты гр. 0381 _____ Странникова Н.
_____ Ефимов Н.
_____ Табаков П.

Преподаватель _____ Заславский М.М.

АННОТАЦИЯ

В рамках данного курса предполагалось разработать какое-либо приложение в команде на одну из поставленных тем. Была выбрана тема построение водных маршрутов по рекам и каналам, так как это отличная возможность разобраться, как работает графовая база данных Neo4j. Найти исходный код и всю дополнительную информацию можно по ссылке: https://github.com/moevm/nosql2h23-river-routes

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 building water routes along rivers and canals was chosen, as this is an excellent opportunity to understand how the Neo4j graph database works. You can find the source code and all additional information at the link: https://github.com/moevm/nosql2h23-river-routes

Оглавление

1. Введение

Цель работы – создать решение для построения речных маршрутов с удобным и понятным интерфейсом для пользователя. Было решено разработать веб-приложение, которое позволит создавать и хранить построенные маршруты.

2. Качественные требования к решению

Требуется разработать приложение с использованием графической СУБД Neo4j.

3. Сценарии использования

Макеты UI

1. Главный экран часть 1, можно перейти к экрану построения маршрута и к списку построенных - см. рис.1

Рисунок 1. Главный экран (часть 1)

2. Главный экран часть 2, представлена информация о проекте - см. рис.2

Рисунок 2. Главный экран (часть 2)

3. Главный экран часть 3, q&a - см. рис.3

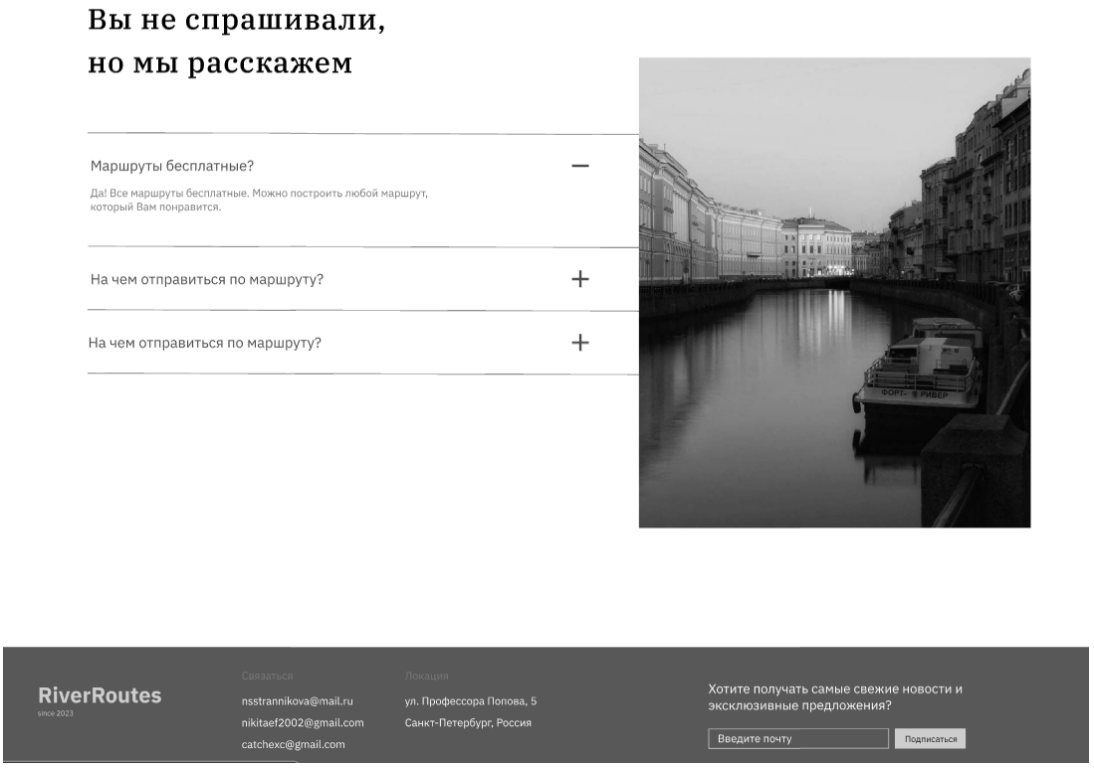


Рисунок 3. Главный экран (часть 3)

4. Экран построения маршрута - см. рис.4

Рисунок 4. Главный экран (часть 4)

5. Экран архива маршрутов - см. рис.5

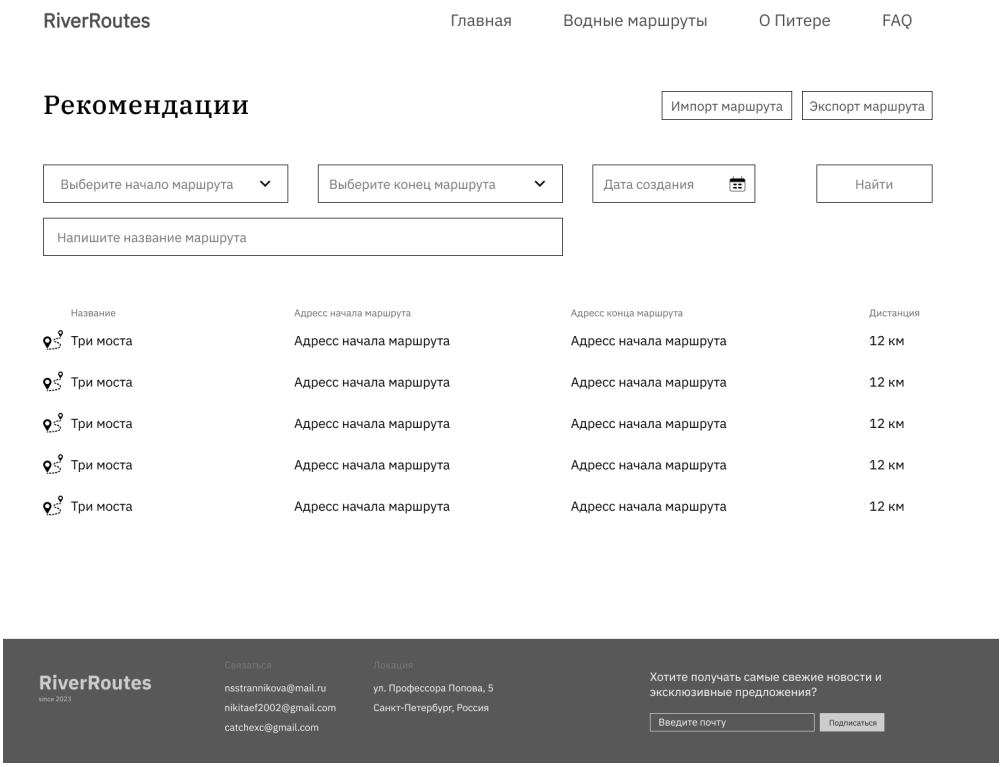


Рисунок 5. Главный экран (часть 5)

6. Экран подробного маршрута - см. рис.6

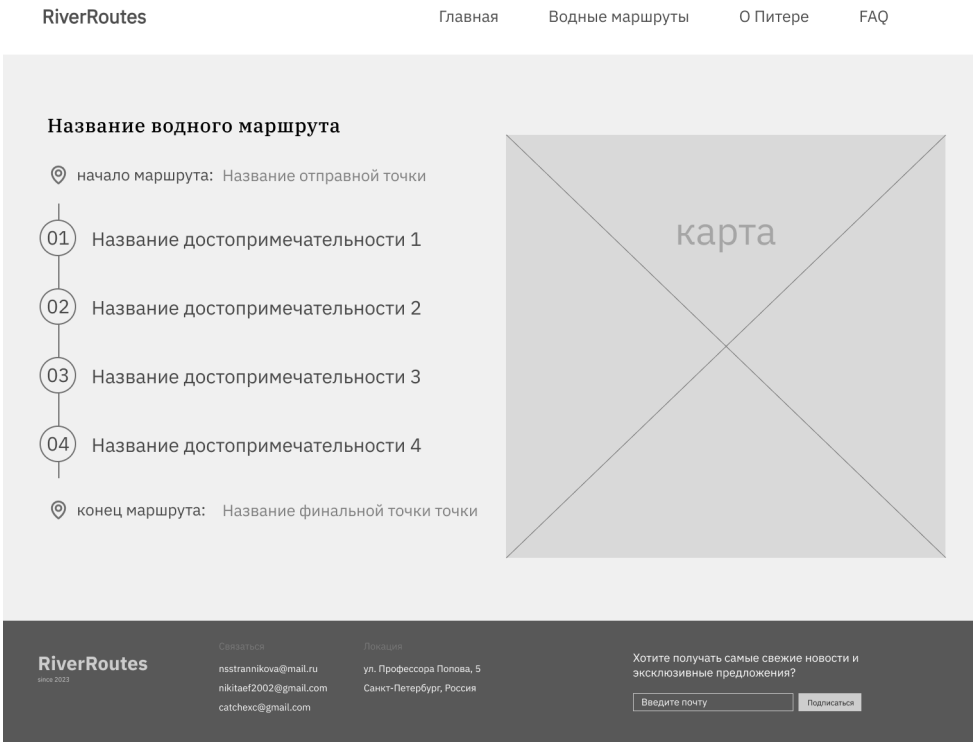


Рисунок 6. Главный экран (часть 6)

Описание сценариев использования

1) *Просмотр подробной информации о водном маршруте*

Действующее лицо: Пользователь

Предусловие: Пользователь находится на экране главной страницы в разделе "Популярные маршруты"

1. Пользовать нажимает мышью на один из блоков с водным маршрутом.
2. Пользователь переходит на экран "Страница с информацией о маршруте".

Результат: Пользователь может просмотреть более подробную информацию о выбранном маршруте.

2) Просмотр часто задаваемых вопросов

Действующее лицо: Пользователь

Предусловие: Пользователь находится на экране главной страницы

1. Пользователь нажимает мышью на кнопку "FAQ" в верхней панеле экрана.
2. Пользователь переходит к разделу "Вы не спрашивали, но мы расскажем".
3. Пользователь нажимает на кнопку с символом плюса рядом с интересующим вопросом.
4. На экране появляется текст с ответом на выбранный вопрос.

Результат: Пользователь может найти ответы на некоторые вопросы, представленные на сайте.

3) Создание своего маршрута

Действующее лицо: Пользователь

Предусловие: Пользователь находится на экране главной страницы

1. Пользователь нажимает кнопку "Выбери свой маршрут".

2. Пользователь переходит на страницу "Страница создания водного маршрута".
3. Пользователь выбирает из списка достопримечательности, которые хочет посетить, нажимая на checkbox.
4. Пользователь выбирает на карте начальную точку маршрута.
5. Пользователь выбирает на карте конечную точку маршрута.
6. Пользователь нажимает кнопку "Построить маршрут".
7. На экране появляется блок с построенным маршрутом на карте и с описанием в текстовом формате.

Результат: Пользователь может построить свой маршрут по выбранным достопримечательностям.

4. Модель данных

Нереляционные модели данных

Схема нереляционной базы данных представлена на рис. 7:

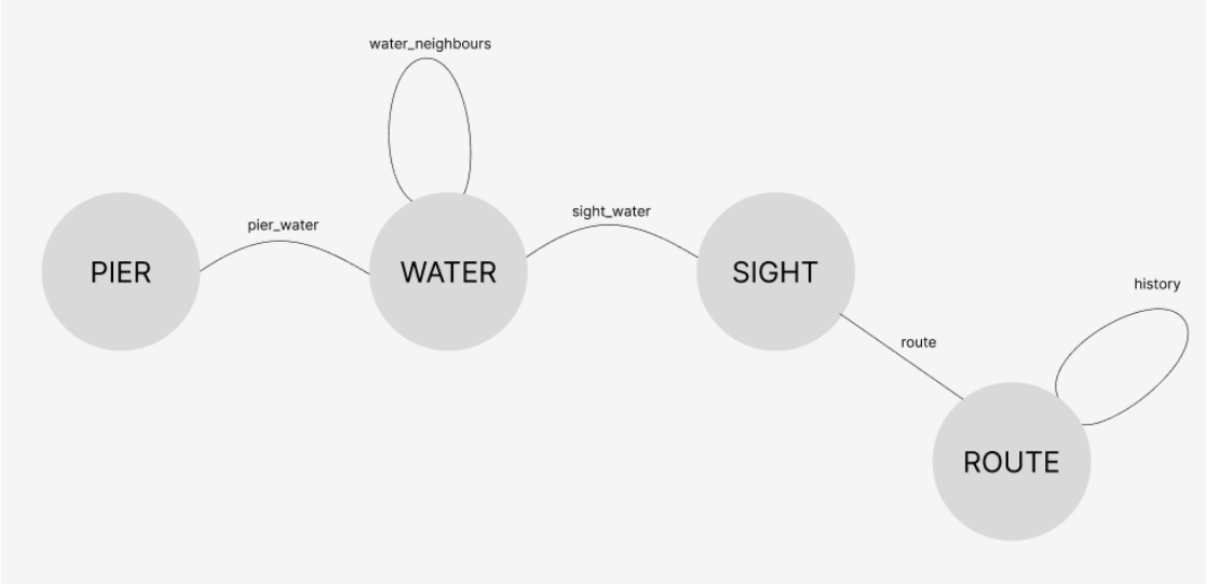


Рисунок 7. Нереляционная модель данных

Оценка удельного объема информации, хранимой в моделях

- Пирс (Pier):
  id - идентификатор пирса. Тип long. V = 8 байт.
  name - название пирса. Тип string. V = 100 байт.
  lat - координата широты пирса. Тип double. V = 8 байт.
  lon - координата долготы пирса. Тип double. V = 8 байт.

updatedAt - дата последнего изменения. Тип Date. V = 8 байт.
Общий объем для одного пирса: 132 байта.

- Водный канал (Water):
  id - идентификатор водного канала. Тип long. V = 8 байт.
  name - название водного канала. Тип string. V = 100 байт.
  lat - координата широты водного канала. Тип double. V = 8 байт.
  lon - координата долготы водного канала. Тип double. V = 8 байт.
  updatedAt - дата последнего изменения. Тип Date. V = 8 байт.
  Общий объем для одного водного канала: 132 байта.
- Достопримечательность (Sight):
  id - идентификатор достопримечательности. Тип long. V = 8 байт.
  name - название достопримечательности. Тип string. V = 100 байт.
  lat - координата широты достопримечательности. Тип double. V = 8 байт.
  lon - координата долготы достопримечательности. Тип double. V = 8 байт.
  wikiLink - ссылка на wiki страницу достопримечательности. Тип string. V = 255 байт.
  updatedAt - дата последнего изменения. Тип Date. V = 8 байт.
  Общий объем для одной достопримечательности: 387 байта.
- Маршрут (Route):
  id - идентификатор маршрута. Тип long. V = 8 байт.
  name - название маршрута. Тип string. V = 100 байт.
  updatedAt - дата последнего изменения. Тип Date. V = 8 байт.
  idPeirStart - id пирса для начала маршрута. Тип long. V = 8 байт.
  idPeirEnd - id пирса для конца маршрута. Тип long. V = 8 байт.
  length - длина маршрута. Тип long. V = 8 байт.
  Общий объём для одного маршрута: 140 байт.
- Связь или ребро, соединяющее два узла:
  id - идентификатор достопримечательности. Тип long. V = 8 байт.
  Общий объем для одной ребра: 8 байта.

От каждого узла могут выходить рёбра, имеющие следующие метки:
PIER_WATER - ребро является связью между пирсом и водным каналом.
SIGHT_WATER - ребро является связью между достопримечательностью и водным каналом.
WATER_NEIGHBORS - ребро является связью между водными каналами.

ROUTE - ребро является связью между маршрутом и начальной и конечной точкой маршрута.

Объем модели от количества маршрутов (включая историю маршрутов):

Vпир пирса ~ 132 байта
Vвод водных каналов ~ 132 байта
Vд достопримечательности ~ 387 байтов
Vм маршрута ~ 140 байт\

V(N) = Vпир * Nпир + Vвод * Nвод + Vд * Nд + Vм * N

Nпир = 100 - количество пирсов
Nвод = 500 - количество водных каналов
Nд = 30 - количество достопримечательностей
V(N) = 88,7 + 0,1 * N Кбайт, где N - кол-во маршрутов.

Избыточность модели

Так как каждая вершина в базе данных будет содержать в себе уникальные координаты, при удалении/сокращении какого-либо элемента для пользователя будет уменьшаться возможность построения разнообразных маршрутов, поэтому чистый объём:
V(N) = 88,7 + 0,1 * N Кбайт

Отношение между фактическим и чистым: 1

Направление роста

По формуле при увеличении кол-ва объектов рост модели линеен.

Примеры запросов

1. Создание пирса:

CREATE (pier:Pier {id: 1, name: 'pier_1', lat: 38.8951, lon: -77.0364, updatedAt: date("2023-11-25") })

2. Создание водного канала:

CREATE (water:Water {id: 1, name: 'water_1', lat: 38.8951, lon: -77.0364, updatedAt: date("2023-11-25") })

3. Создание достопримечательности:

CREATE (sight:Sight {id: 1, name: 'sight_1', lat: 38.8951, lon: -77.0364, wikiLink: "https://wikilink_1.ru", updatedAt: date("2023-11-25") })

4. Соединение двух водных каналов:

MATCH(water_1:Water {id: 1})
MATCH(water_2:Water {id: 2})
CREATE (water_1)-[:WATER_NEIGHBOURS]->(water_2)

5. Создание маршрута:

CREATE (route:Route {id: 1, name: 'route_1', updatedAt: date("2023-11-25"), idPeirStart: 1, idPeirEnd: 3, length: 4568 })

6. Изменение названия маршрута:

MATCH(r:Route {id: 1}) SET r.name = 'new_name' RETURN r

7. Удаление маршрута:

MATCH(r:Route {id: 1})
DETACH DELETE r

8. Поиск кратчайшего маршрута между двумя точками (старт и финиш).

MATCH path = shortestPath((startNode:Label {id: 'start_id'})-[..10]-(endNode:Label {id: 'end_id'}))*
RETURN path;

Аналог модели данных для SQL СУБД

Схема нереляционной базы данных представлена на рис. 8:

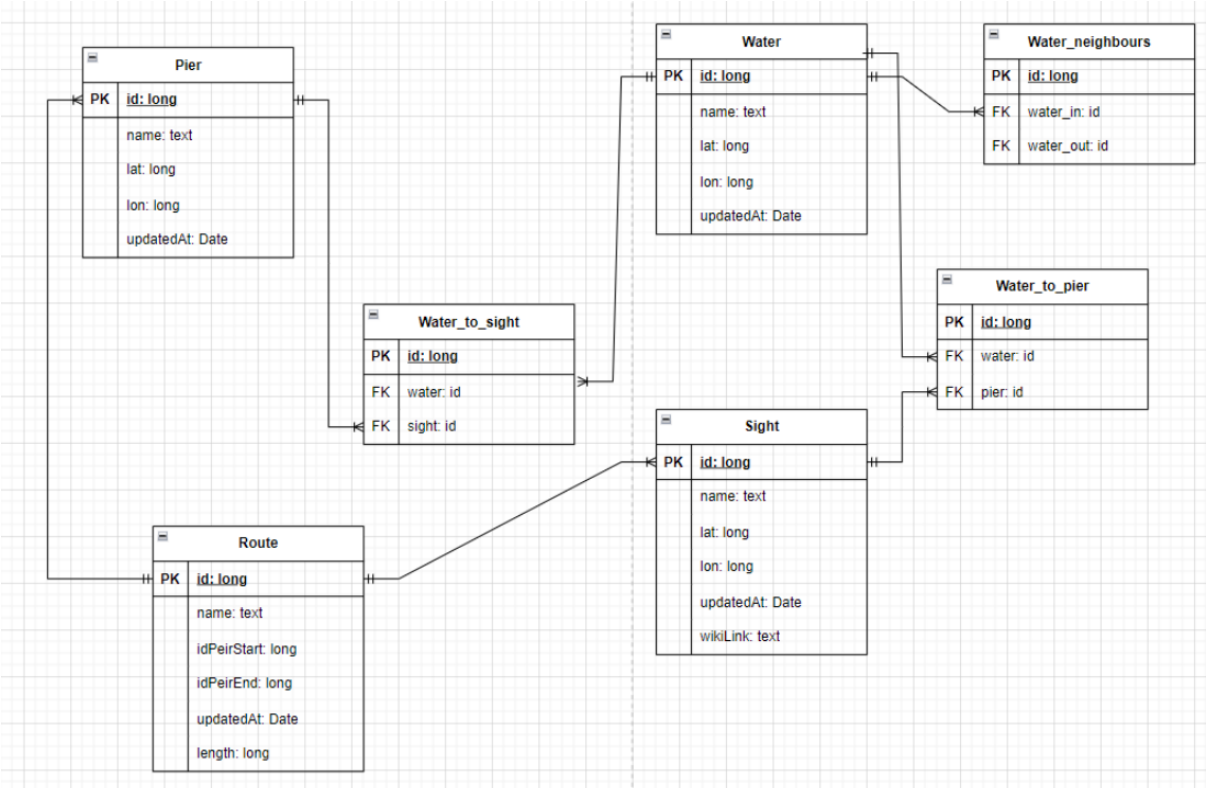


Рисунок 8. Реляционная модель данных

Оценка удельного объема информации, хранимой в моделях

- Пирс (Pier):
  id - идентификатор пирса. Тип long. V = 8 байт.
  name - название пирса. Тип text. V = 100 байт.
  lat - координата широты пирса. Тип double. V = 8 байт.
  lon - координата долготы пирса. Тип double. V = 8 байт.
  updatedAt - дата последнего изменения. Тип Date. V = 8 байт.
  Общий объем для одного пирса: 132 байта.
- Водный канал (Water):
  id - идентификатор водного канала. Тип long. V = 8 байт.
  name - название водного канала. Тип text. V = 100 байт.
  lat - координата широты водного канала. Тип double. V = 8 байт.
  lon - координата долготы водного канала. Тип double. V = 8 байт.
  updatedAt - дата последнего изменения. Тип Date. V = 8 байт.
  Общий объем для одного водного канала: 132 байта.
- Достопримечательность (Sight):
  id - идентификатор достопримечательности. Тип long. V = 8 байт.
  name - название достопримечательности. Тип text. V = 100 байт.
  lat - координата широты достопримечательности. Тип double. V = 8

байт.
lon - координата долготы достопримечательности. Тип double. V = 8 байт.
wikiLink - ссылка на wiki страницу достопримечательности. Тип string. V = 255 байт.
updatedAt - дата последнего изменения. Тип Date. V = 8 байт.
Общий объем для одной достопримечательности: 387 байта.

- Маршрут (Route):
  id - идентификатор маршрута. Тип long. V = 8 байт.
  name - название маршрута. Тип text. V = 100 байт.
  updatedAt - дата последнего изменения. Тип Date. V = 8 байт.
  length - длина маршрута. Тип long. V = 8 байт.
  Общий объём для одного маршрута: 124 байт.
- Граничащие водные каналы (Water_neighbours):
  id - идентификатор границы водных каналов. Тип long. V = 8 байт.
  water_in_id - идентификатор входящего водного канала. Тип long. V = 8 байт.
  water_out_id - идентификатор исходящего водного канала. Тип long. V = 8 байт.
  Общий объём для одного маршрута: 24 байт.
- Граничащие пирсы с водными каналами (Water_to_pier):
  id - идентификатор границы пирса с водным каналом. Тип long. V = 8 байт.
  water_id - идентификатор водного канала. Тип long. V = 8 байт.
  pier_id - идентификатор пирса. Тип long. V = 8 байт.
  Общий объём для одного маршрута: 24 байт.
- Граничащие достопримечательности с водным каналом (Water_to_sight):
  id - идентификатор границы достопримечательности с водным каналом. Тип long. V = 8 байт.
  water_id - идентификатор водного канала. Тип long. V = 8 байт.
  sight_id - идентификатор достопримечательности. Тип long. V = 8 байт.
  Общий объём для одного маршрута: 24 байт.
- Маршрут (Route): id - идентификатор маршрута. Тип long. V = 8 байт.
  name - название маршрута. Тип string. V = 100 байт.

updatedAt - дата последнего изменения. Тип Date. V = 8 байт.
idPeirStart - id пирса для начала маршрута. Тип long. V = 8 байт.
idPeirEnd - id пирса для конца маршрута. Тип long. V = 8 байт.
length - длина маршрута. Тип long. V = 8 байт.
Общий объём для одного маршрута: 140 байт.

Объем модели от количества маршрутов (включая историю маршрутов):

Vпир пирса ~ 132 байта
Vвод водных каналов ~ 132 байта
Vд достопримечательности ~ 387 байтов
Vм маршрута ~ 140 байт
Vг.в. граничащих водных каналов ~ 24 байта
Vг.д. граничащих водных каналов и достопримечательностей ~ 24 байта
Vг.п. граничащих водных каналов и пирсов ~ 24 байта\

V(N) = (Vпир * Nпир + Nпир * Vг.п.) + (Vвод * Nвод + Nв/2 * Vг.в.) + (Vд * Nд + Nд * Vг.д.) + Vм * N

Nпир = 100 - количество пирсов
Nвод = 500 - количество водных каналов
Nд = 30 - количество достопримечательностей
V(N) = 98,7 + 0,1 * N Кбайт, где N - кол-во маршрутов.

Избыточность модели

В данной модели избыточны таблицы с граничащими объектами: Water_neighbours, Water_to_pier, Water_to_sight.

Тогда: V(N) = 88,7 + 0,1 * N Кбайт

Отношение между фактическим и чистым: 1,11

Направление роста

По формуле при увеличении кол-ва объектов рост модели линеен.

Примеры запросов

1. Создание пирса:

INSERT INTO Pier (name, lat, lon, updatedAt)

VALUE ('pier_1', 38.8951, -77.0364, '2023-11-25');

2. Создание водного канала:

INSERT INTO Water (name, lat, lon, updatedAt)

VALUE ('water_1', 38.8951, -77.0364, '2023-11-25');

3. Создание достопримечательности:

INSERT INTO Pier (name, lat, lon, wikiLink, updatedAt)

VALUE ('sight_1', 38.8951, -77.0364, 'https://wikilink_1.ru', '2023-11-25');

4. Соединение двух водных каналов:

INSERT INTO Water_to_water (water_in_id, water_out_id)

VALUE (1, 2);

5. Создание маршрута:

INSERT INTO Route (name, lat, lon, idPeirStart, idPeirEnd, updatedAt)

VALUE ('sight_1', 38.8951, -77.0364, 1, 3, '2023-11-25');

6. Изменение названия маршрута:

UPDATE Route SET name = 'new_name' WHERE id = 1;

7. Удаление маршрута:

DELETE FROM Route WHERE id = 1;

8. Поиск кратчайшего маршрута между двумя точками (старт и финиш).

Данный запрос невозможно сделать, так как на уровне субд невозможно найти кратчайший путь.

Сравнение моделей

NoSQL требует заметно меньше памяти, сравнивая объем данных в модели NoSQL и объем данных в модели SQL.

Вывод

NoSQL подходит для данной задачи лучше чем SQL, так как замечен очевидный выигрыш по памяти.

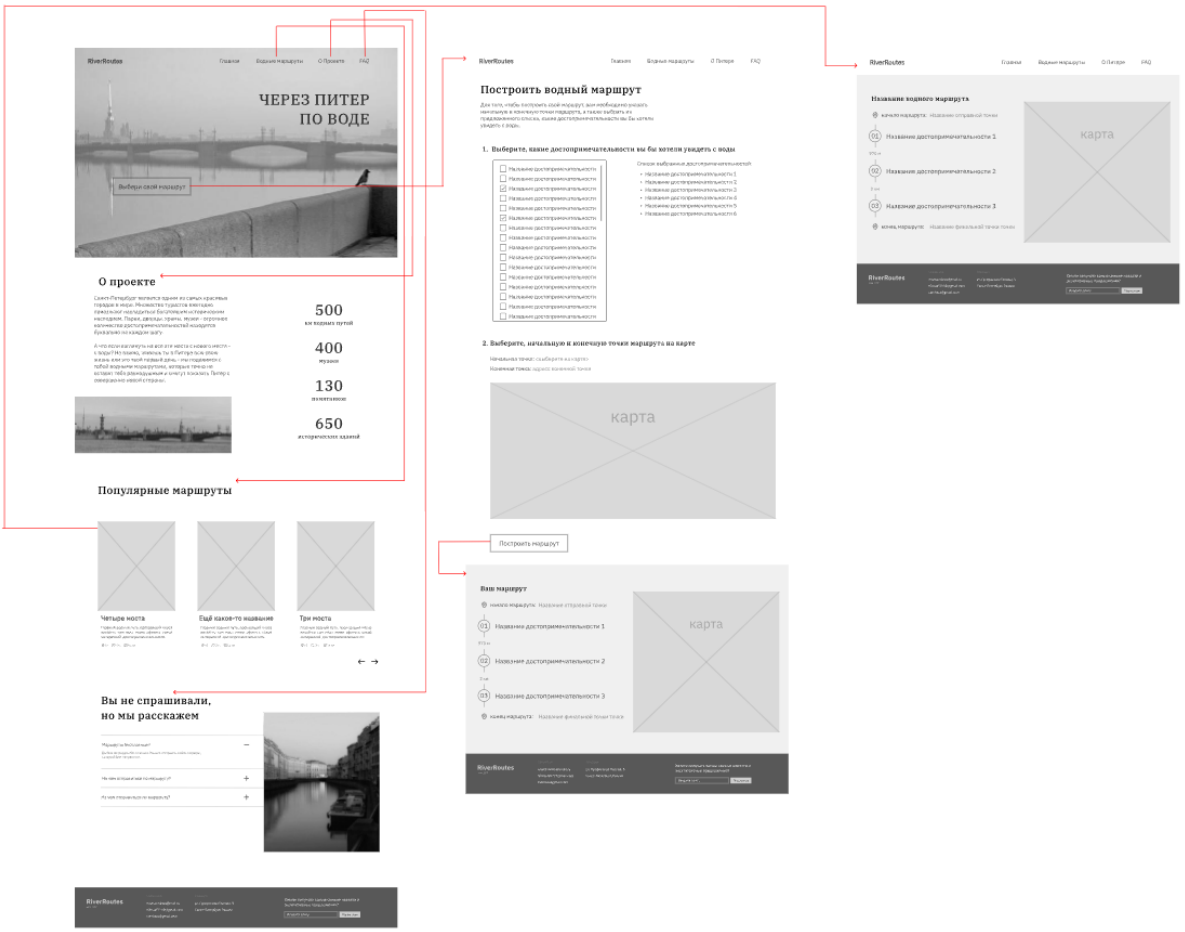
5. Разработанное приложение

Краткое описание

Back-end представляет из себя Java-приложение. При запуске приложения происходит запрос к OpenStreetMap сервису, который возвращает данные о реках и канала определенной области. Далее происходит процесс обработки данных, связывания их между собой и сохранения в графовую бд. Далее построение маршрута и любые операции с данными происходят через запросы к бд, которые реализованы в сервисе работы с базой данных.

Front-end – это web-приложение, которое использует API back-end приложения.

Схема экранов приложения



Использованные технологии

База данных: Neo4j

Back-end: Spring Boot

Front-end: React

Ссылки на Приложение

Ссылка на github: https://github.com/moevm/nosql2h23-river-routes

6. Вывод

Результаты

В ходе работы было разработано приложение построения водных маршрутов по Неве, используя данные взятые с Open Street Map сервиса. Приложение позволяет строить маршруты и сохранять их.

Недостатки и пути для улучшения полученного решения

На данный момент процесс получения данных через OSM, их обработка и сохранения в базу данных требует больших вычислительных

мощностей компьютера. По этой причине в данной работе рассматривается только одна небольшая область, чтобы процесс обработки не затягивался на многие часы. В будущем это процесс нужно автоматизировать.

Также уличить можно и сам функционал приложения. Добавить больше информации о достопримечательностях и тп.

Будущее развитие решения

Планируется расширение функционала приложения.

7. Приложения

Документация по сборке и развертыванию приложения

1. Для запуска контейнеров: переходим в nosql2h23-river-routes и пишем команду: docker-compose up --build
2. переходим на localhost:8080

8. Используемая литература

1. Документация к Neo4j: https://neo4j.com/docs/
2. Документация к Spring Boot: https://docs.spring.io/spring-boot/docs/current/reference/html/
3. Документация к React: https://ru.legacy.reactjs.org/docs/getting-started.html