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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные системы управления базами данных»

Тема: Информационная система аренды электросамокатов

Студентка гр. 9382 _____ Сорочина М.В.

Студент гр. 9382 _____ Рыжих Р.В.

Студент гр. 9382 _____ Юрьев С.Ю.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2022

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)

Студенты

Сорочина М.В.

Рыжих Р.В.

Юрьев С.Ю.

Группа 9382

Тема работы: Информационная система аренды электросамокатов

Исходные данные:

Необходимо реализовать приложение аренды самокатов при помощи СУБД neo4j

Содержание пояснительной записки:

«Содержание»

«Введение»

«Качественные требования к решению»

«Сценарии использования»

«Модель данных»

«Разработанное приложение»

«Заключение»

«Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 05.09.2022

Дата сдачи реферата: 16.12.2022

Дата защиты реферата: 16.12.2022

Студентка гр. 9382 _____ Сорочина М.В.

Студент гр. 9382 _____ Рыжих Р.В.

Студент гр. 9382 _____ Юрьев С.Ю.

Преподаватель _____ Заславский М.М.

АННОТАЦИЯ

Была разработана информационная система аренды электросамокатов. При разработке использовались neo4j и express. Создан прототип, который включает в себя просмотр базы данных с помощью таблиц с поддержкой фильтрации по полям, добавление и редактирование информации о самокатах, импорт и экспорт данных в формате json. Запуск приложения реализован при помощи docker-compose build --no-cache.

SUMMARY

An information system for renting electric scooters was developed. The development used neo4j and express. A prototype has been created, which includes browsing the database using tables with support for filtering by fields, adding and editing information about scooters, importing and exporting data in json format. Application launch is implemented using docker-compose build –no-cache.

СОДЕРЖАНИЕ

	Введение	6
1.	Качественные требования к решению	7
2.	Сценарии использования	8
2.1.	Макет пользовательского интерфейса	8
2.2.	Сценарии использования	8
2.3.	Преобладающие операции.	12
3.	Модель данных	13
3.1.	Нереляционная модель данных	13
3.2.	Аналог модели данных для SQL СУБД	18
3.3.	Сравнение моделей	21
3.4.	Вывод	22
4.	Разработанное приложение	23
4.1.	Краткое описание	23
4.2.	Схема экранов приложения	23
4.3.	Использованные технологии	25
	Выводы	26
	Список использованных источников	27
	Приложение А. Название приложения	28

ВВЕДЕНИЕ

Цель работы — создать веб приложение для аренды электросамокатов.

Задачи:

1. Сформулировать основные сценарии использования и составить макет
2. Разработать модель данных
3. Разработать прототип «Хранение и представление»
4. Разработать прототип «Анализ»

1. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ

Текущие требования к решению:

1. Релизовать страницу логина и добавить хотя бы по одному пользователю на каждую из ролей

2. Просмотр базы данных с помощью таблиц

3. Добавление новых самокатов и изменение информации о существующих

4. Фильтрация данных

5. Приложение разворачивается через docker-compose build --no-cache из репозитория на ubuntu

6. Импорт и экспорт данных в формате json

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет пользовательского интерфейса

Макет представлен в приложении А.

2.2. Сценарии использования

Сценарии использования для пользователя.

Сценарий использования “Аутентификация”

Основной сценарий:

1) Действующее лицо вводит свои логин и пароль

2) Нажимает кнопку войти

3) Успешный вход в систему, переход на страницу с картой, на которой отмечены самокаты

Альтернативный сценарий:

1) Действующее лицо вводит свои логин и пароль

2) Нажимает кнопку войти

3) Данные введены неверно, остается на этой же странице, переходит к шагу 1

Сценарий использования “Выход”

Основной сценарий:

1) Действующее лицо нажимает кнопку выхода в правом верхнем углу

2) Оказыается на странице аутентификации

Сценарий использования “Просмотр информации о самокате”

Основной сценарий:

1) Пользователь выбирает одну из точек на карте, которой отмечен самокат

2) Нажимает

3) Появляются информация о самокате: его номер и процент заряда, а также варианты “взять” и “отмена”

Сценарий использования “Аренда самоката”

Основной сценарий:

1-3) То же, что в “Просмотр информации о самокате”

4) Нажимает кнопку взять

5) Начинается поездка, на экране есть информация о взятом самокате, время и сумма поездки

Сценарий использования “Завершение поездки”

Основной сценарий:

1) При условии, что пользователь уже взял самокат: нажать “Закончить поездку”

2) Поездка окончена, карта принимает изначальный вид (рис. 2)

Сценарий использования “Просмотр тарифов/правил”

Основной сценарий:

1) Нажать на кнопку меню в левом верхнем углу

2) Выбрать вариант тарифы/правила

3) Происходит переход на страницу, где описаны тарифы/правила

Сценарий использования “Поиск по адресу”

Основной сценарий:

1) Пользователь вводит адрес в поисковую строку

2) Выбирает из предложенных

3) Происходит перемещение и масштабирование карты так, чтобы в центре был выбранный адрес

Сценарий использования “Просмотр профиля”

Основной сценарий:

1) Пользователь открывает боковое меню

2) Выбирает пункт “профиль”

Сценарий использования “Просмотр истории поездок”

Основной сценарий:

1) Пользователь открывает боковое меню

2) Выбирает пункт “профиль”. Там сразу представлена вся история поездок. Ее можно отфильтровать по датам, отсортировать по дате/времени/сумме/времени старта

Сценарии использования для работника.

Работник имеет те же возможности, что пользователь + те, что будут описаны ниже

Сценарий использования “Добавление самоката или его редактирование”

Основной сценарий:

1) Работник открывает меню (кнопка в левом верхнем углу)

2) Выбирает пункт “Добавить/редактировать самокат”

3) Происходит переход на страницу добавления и редактирования

4) Заполняет поля

5) Нажимает “Добавить/редактировать”

Сценарий использования “Выгрузка на площадку”

Основной сценарий:

1) Работник открывает меню (кнопка в левом верхнем углу)

2) Выбирает пункт “Выгрузка на место”

3) Происходит переход на страницу добавления

4) Заполняет поля “№ самоката” и выбирает площадку, на которую выгружает

5) Нажимает “Подтвердить”

Сценарий использования “Отгрузка на склад”

Основной сценарий:

1) Работник открывает меню (кнопка в левом верхнем углу)

2) Выбирает пункт “Отгрузка на склад”

3) Происходит переход на страницу добавления

4) Заполняет поля “№ самоката” и выбирает склад, на который отгрузит самокаты

5) Нажимает “Подтвердить”

Сценарий использования “Просмотр БД клиентов/самокатов/складов/площадок выгрузки/поездок”

Основной сценарий:

1) Работник открывает меню (кнопка в левом верхнем углу)

2) Выбирает пункт “Базы данных”

3) Происходит переход на страницу с выбором бд и поиском по бд

4) Выбирает пункт “Клиенты”/“Самокаты”/“Склады”/“Площадки выгрузки”/“Поездки”

5) Происходит переход на страницу, где представлена таблица со всеми даныыми о выбранном пункте (Примеры таблиц на рис. 6 макета работников)

Сценарий использования “Просмотр фильтрованных данных из баз данных”

Основной сценарий:

1) Работник открывает меню (кнопка в левом верхнем углу)

2) Выбирает пункт “Базы данных”

3) Происходит переход на страницу с выбором бд и поиском по бд

4) Работник выбирает необходимые ему данные. Например, определенный самокат и клиента

5) Нажимает “Поиск”

6) Появляется таблица, которая содержит в себе найденные данные. Например, все записи, где есть выбранный самокат и клиент.

Сценарий использования “Просмотр агрегированных данных”

Основной сценарий:

1) Работник открывает меню (кнопка в левом верхнем углу)

2) Выбирает пункт “Базы данных”

3) Происходит переход на страницу с выбором бд, поиском по бд и аггрегориванными данными

4) Работник выбирает необходимые ему агрегированные данные и нажимает кнопку подтверждения

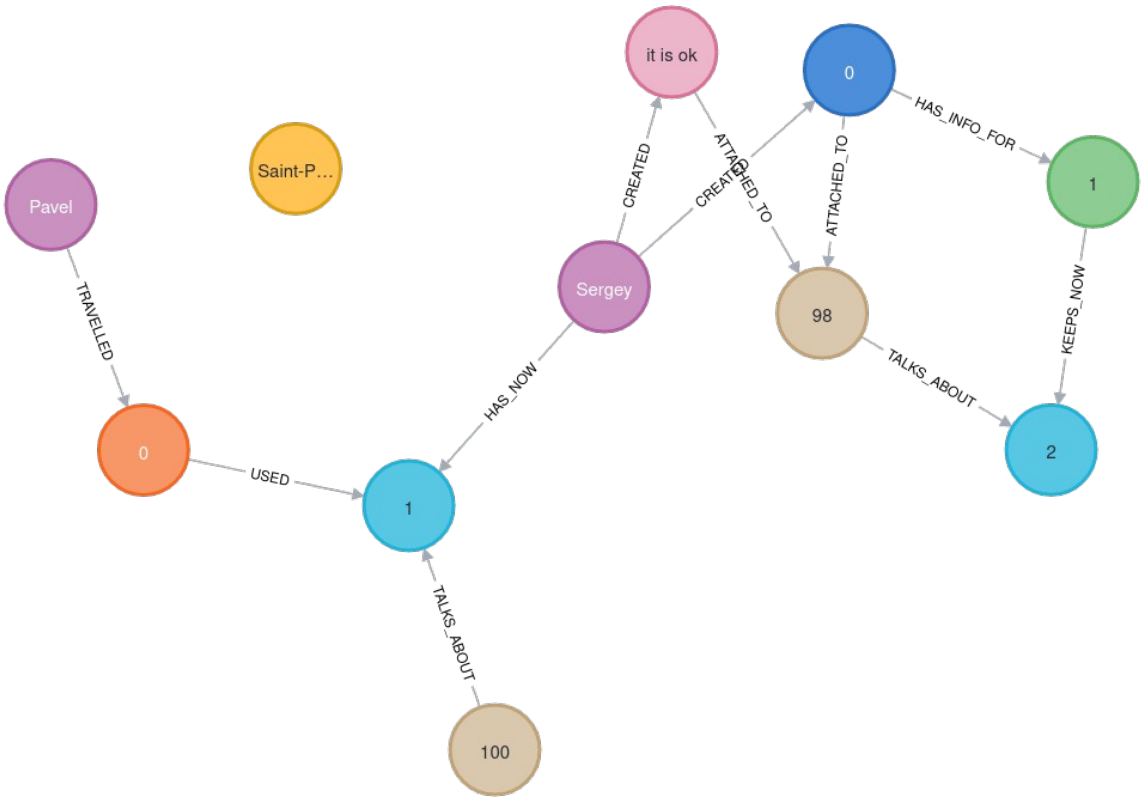
5) Происходит переход на страницу с выбранными данными

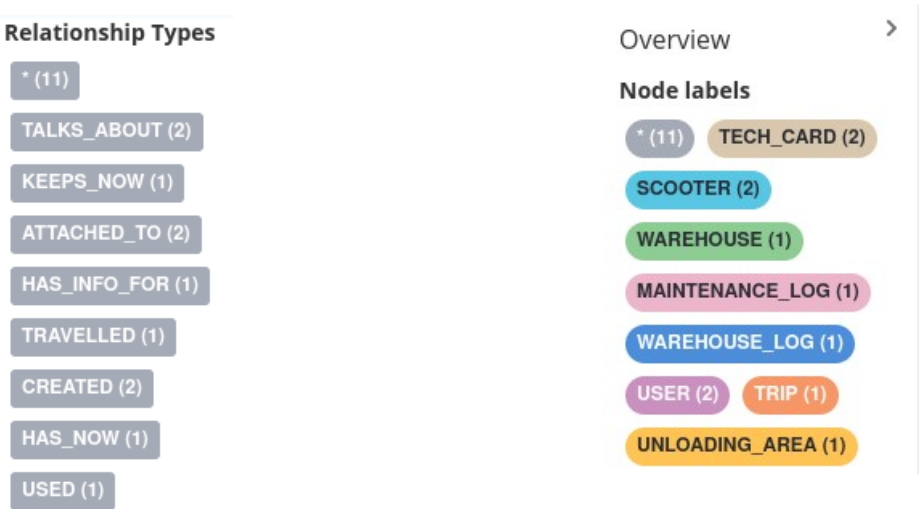
2.3. Преобладающие операции.

Преобладать будут операции чтения

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных





Список сущностей модели:

1. USER – информация о пользователе

    * name – имя (String - 50*2 байта)

    * phone – номер телефона (String - 15*2 байт)

    * login – логин (String - 50*2 байта)

    * password – пароль (String - 50*2 байта)

    * type – тип пользователя (byte - 1 байт)

    Итого: 181 байт

2. TRIP – информация о поездке

    * time_start – время начала поездки (long – 8 байт)

    * time_end – время конца поездки (long – 8 байт)

    * cost – итоговая стоимость поездки (int – 4 байта)

    * status – текущее состояние поездки (byte - 1 байт)

    Итого: 21 байт

3. SCOOTER – информация о состоянии самоката

    * number – уникальный номер самоката (byte - 1 байт)

    * battery – процент зарядки (byte - 1 байт)

    * coordinate_x – координата местоположения (double – 8 байт)

    * coordinate_y – координата местоположения (double – 8 байт)

    * status – текущее статус задействованности самоката (byte - 1 байт)

    Итого: 19 байт

4. TECH_CARD – техническая информация о конкретном самокате

    * creationYear – год создания самоката (int – 4 байта)

    * manufacturer – производитель (String - 50*2 байта)

    * maxPowerCapacity – максимальная мощность самоката (int – 4 байта)

    * mileage – километраж (int – 4 байта)

    Итого: 112 байт

5. WAREHOUSE – информация о складе самокатов

    * numberOfWarehouse – номер склада (byte - 1 байт)

    Итого: 1 байт

6. UNLOADING_AREA – информация о расположении площадки выгрузки самокатов

* coordinate_x – координата местоположения (double – 8 байт)

* coordinate_y – координата местоположения (double – 8 байт)

* address – адрес (String - 50*2 байта)

Итого: 116 байт

7. WAREHOUSE_LOG – информация о добавлении/удалении самокатов со склада

* action_type – тип активности (добавление/удаление) (byte - 1 байт)

* creation_time — момент создания лога (long – 8 байт)

Итого: 9 байт

8. MAINTENANCE_LOG – информация о результатах техобслуживании конкретного самоката

* message – информация по результатам ТО (String - 50*2 байта)

Итого: 100 байт

В итоге всего 558 байт

Все связи между сущностями не имели полей, поэтому примем их размер за 0 байт

Пусть количество USER = A, количество SCOOTER и TECH_CARD = B, количество TRIP = C, количество WAREHOUSE = D, количество UNLOADING_AREA = E, количество WAREHOUSE_LOG и MAINTENANCE_LOG = F.

Тогда общий объем данных будет = 181A + 131B + 21C + D + 116E + 109F

А чистый будет = 181A + 131B + 20C + D + 116E + 109F

Выразим соотношения через количество поездок. В-среднем на один самокат придется 15 поездок, на одного пользователя - 1/3 самоката (пользователей больше, чем самокатов, т.к. люди катаются в разное время), на

один склад приходится 30 самокатов, на одну площадку выгрузки приходится по 5 самокатов, а на каждые 10 поездок приходится 1 ТО.

Итого получаем A=5C, B=15C, D=0.5C, E=3C, F=0.1C

Тогда избыточность модели будет: 3250.4C/3249.4C = 1.0003

Увеличение количества объектов любой из сущностей приведет лишь к линейному росту

Запросы к модели с помощью которых реализуются сценарии использования

* Запрос на поиск пользователя с соответствующим логином и паролем:
```
MATCH (user:USER {login:enteredLogin, password:enteredPassword})
RETURN user;
```

* Получить информацию о доступных самокатах
```
MATCH (sc:SCOOTER {status=1})
RETURN sc.coordinate_x, sc.coordinate_y, sc.battery, sc.number;
```

* Арендовать выбранный самокат
```
MATCH (sc:SCOOTER {number:chosenOne})
MATCH (user:USER {login:currLogin})
CREATE (trip:TRIP {time_start:currSecond})
CREATE (trip)-[:USED]→(sc)
CREATE (user)-[:TRAVELLED]->(trip)
SET sc.status=2, trip.status=1
RETURN trip.time_start, sc
```

* Завершить поездку на выбранном самокате
```
MATCH (sc:SCOOTER {number:chosenOne})
MATCH (user:USER {login:currLogin})
MATCH (trip:TRIP {time_start:currStartTime})←[:TRAVELLED]-(user)
SET trip.time_end=currSecond, trip.status=0, trip.cost=value, sc.status=1
```

* Поиск по адресу
```
MATCH (sc:SCOOTER {status=1})
RETURN sc.coordinate_x, sc.coordinate_y, sc.battery, sc.number;
```

* Добавление/редактирование самоката
```
MATCH (user:USER {login:currLogin})
MERGE (sc:SCOOTER {status:4, number:enteredNumber})
MERGE (user)-[:HAS_NOW]->(sc)
```

```
MERGE (techCard:TECH_CARD)-[:TALKS_ABOUT]->(sc)
SET techCard.creationYear=currYear,  techCard.manufacturer=enteredManufacturer,
techCard.maxPowerCapacity=enteredValue, techCard.mileage=0
CREATE (logM:MAINTENANCE_LOG {message:"text"})
CREATE (logW:WAREHOUSE_LOG {action_type:1,  creation_time:currTime})
CREATE (logM)-[:ATTACHED_TO]->(techCard)
CREATE (logW)-[:ATTACHED_TO]->(techCard)
CREATE (user)-[:CREATED]->(logM)
CREATE (user)-[:CREATED]->(logW)
```

* Выгрузка на площадку

```
MATCH (area:UNLOADING_AREA {address:chosenAddress})
MATCH (sc:SCOOTER {status:4})
RETURN area
```

* для всех выбранных скутеров выполнить:

```
MATCH (sc:SCOOTER {number:currNumber})←[pointer:HAS_NOW]-
(user:USER {login:currLogin})
SET sc.coordinate_x=area.coordinate_x, sc.coordinate_y=area.coordinate_y,
sc.status=1, sc.battery=100
DELETE  pointer
```

* Фильтры поиска:

```
MATCH (clients:USER)
RETURN clients
MATCH (client:USER {name:chosenName})
RETURN client

MATCH (sc:SCOOTER)
RETURN sc
MATCH (sc:SCOOTER {number:chosenNumber})
RETURN sc

MATCH (wh:WAREHOUSE)
RETURN wh
MATCH (wh:WAREHOUSE {numberOfWarehouse:chosenNumber})
RETURN wh

MATCH (areas:UNLOADING_AREA)
RETURN areas
MATCH (areas:UNLOADING_AREA {address:chosenAddress})
RETURN areas
```

* пользователи у которых давно не было поездок

```
MATCH (users: USER) -[:TRAVELLED]→(trip:TRIP)
WHERE trip.time_end < ourConstValue
RETURN users
LIMIT 5
```

```
* самые далекие самокаты
MATCH (sc: SCOOTER)
RETURN sc
ORDER BY sc.coordinate_x DESC
LIMIT 5
```

Количество запросов для совершения юзкейсов не зависит от числа объектов в БД и прочих параметров

Не было задействовано никаких дополнительных коллекций

3.2. Аналог модели данных для SQL СУБД

Список сущностей модели:

1. USERS – таблица с пользователями
    * id – первичный ключ (INT(10) – 4 байта)
    * name – имя (CHAR(50) - 50 байт)
    * phone – номер телефона (CHAR(15) - 15 байт)
    * login – логин (CHAR(50) - 50 байт)
    * password – пароль (CHAR(50) - 50 байт)
    * type – тип пользователя (TINYINT(3) - 1 байт)

    Итого: 170 байт
2. TRIPS – таблица с поездками
    * id – первичный ключ (INT(10) – 4 байта)
    * time_start – время начала поездки (BIGINT(10) – 8 байт)
    * time_end – время конца поездки (BIGINT(10) – 8 байт)
    * cost – итоговая стоимость поездки (INT(10) – 4 байта)
    * status – текущее состояние поездки (TINYINT(3) - 1 байт)
    * ID_scooter — внешний ключ (INT(10) – 4 байта)
    * ID_user — внешний ключ (INT(10) – 4 байта)

Итого: 33 байт

3. SCOOTERS – таблица с информацией о состояниях самокатов

* id – первичный ключ (INT(10) – 4 байта)

* ID_worker – внешний ключ (INT(10) – 4 байта)

* number – уникальный номер самоката (TINYINT(3) - 1 байт)

* battery – процент зарядки (TINYINT(3) - 1 байт)

* coordinate_x – координата местоположения (BIGINT(10) – 8 байт)

* coordinate_y – координата местоположения (BIGINT(10) – 8 байт)

* status – текущее статус задействованности самоката (TINYINT(3) - 1 байт)

Итого: 27 байт

4. TECH_CARDS – техническая информация о конкретном самокате

* id – первичный ключ (INT(10) – 4 байта)

* ID_scooter — внешний ключ (INT(10) – 4 байта)

* creationYear – год создания самоката (INT(10) – 4 байта)

* manufacturer – производитель (CHAR(50) – 50 байт)

* maxPowerCapacity – максимальная мощность самоката (INT(10) – 4 байта)

* mileage – километраж (INT(10) – 4 байта)

Итого: 70 байт

5. WAREHOUSES – информация о складе самокатов

* id – первичный ключ (INT(10) – 4 байта)

* ID_scooter — внешний ключ (INT(10) – 4 байта)

* numberOfWarehouse – номер склада (TINYINT(3) - 1 байт)

Итого: 9 байт

6. UNLOADING_AREAS – информация о расположении площадки выгрузки самокатов

* id – первичный ключ (INT(10) – 4 байта)

* coordinate_x – координата местоположения (BIGINT(10) – 8 байт)

* coordinate_y – координата местоположения (BIGINT(10) – 8 байт)

* address – адрес (CHAR(50) – 50 байт)

Итого: 70 байт

7. WAREHOUSE_LOGS – информация о добавлении/удалении самокатов со склада

* id – первичный ключ (INT(10) – 4 байта)

* ID_worker — внешний ключ (INT(10) – 4 байта)

* ID_warehouse — внешний ключ (INT(10) – 4 байта)

* ID_techCard — внешний ключ (INT(10) – 4 байта)

* action_type – тип активности (добавление/удаление) (TINYINT(3) - 1 байт)

* creation_time — момент создания лога (BIGINT(10) – 8 байт)

Итого: 21 байт

8. MAINTENANCE_LOGS – информация о результатах техобслуживании конкретного самоката

* id – первичный ключ (INT(10) – 4 байта)

* ID_worker — внешний ключ (INT(10) – 4 байта)

* ID_scooter — внешний ключ (INT(10) – 4 байта)

* message – информация по результатам ТО (CHAR(50) – 50 байт)

Итого: 62 байт

В итоге всего: 474 байт

Общий объем данных будет = 170A + 97B + 33C + 21D + 70E + 83F

А чистый будет = 166A + 81B + 21C + 9D + 66E + 59F

Избыточность модели будет: 2566.8C/2274.4C = 1.1286

Увеличение количества объектов в любой из таблиц приведет лишь к линейному росту

Запросы:

* Вход

SELECT * FROM users WHERE login=enteredValue AND password=enteredPassword

* Получить информацию о доступных самокатах

SELECT * FROM scooters WHERE status=1

* Арендовать выбранный самокат

INSERT INTO trips VALUES (‘currTime’, ‘0’, ‘0’, ’1’, ‘currScooterID’, ‘currUserID’)

UPDATE scooters

SET status=2

WHERE status=1;

* Завершить поездку на выбранном самокате

UPDATE trips

SET status=0, timeEnd=currTime, cost=value

WHERE status=1;

UPDATE scooters

SET status=1

WHERE status=2;

* Поиск по адресу

SELECT * FROM scooters WHERE status=1

3.3. Сравнение моделей

По занимаемой памяти Neo4j уступает SQL, но у Neo4j благодаря ее структуре почти нулевая избыточность. В примере разница в памяти обеспечивается тем, что в sql тип CHAR занимает 1 байт, а в Neo4j – два. Если не учитывать при подсчете CHAR, то SQL занимает больше места, чем Neo4j,

это связано с тем, что SQL имеет большие накладные расходы из-за дублирования информации

3.4. Вывод

SQL имеет большое дублирование информации, больший расход места (если не учитывать CHAR) и большее среднее время отклика, но зато он обычно требует меньшее количество запросов по сравнению с Neo4j

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

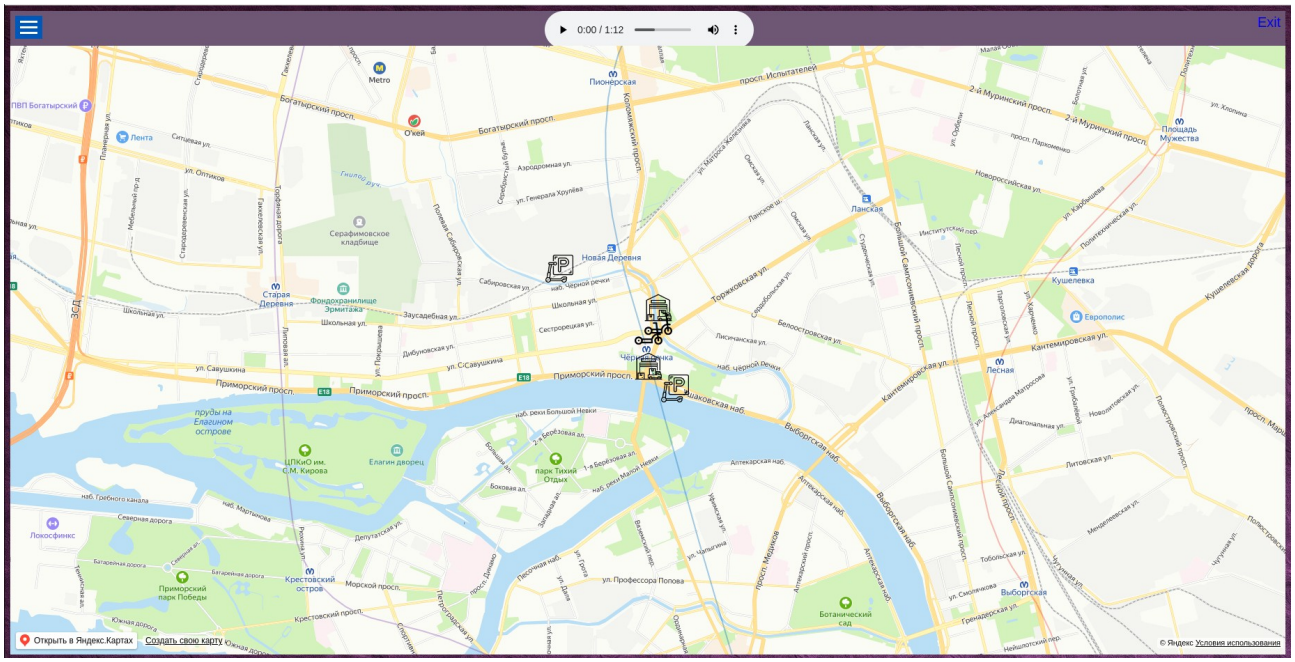
4.1. Краткое описание

При входе пользователь получает определенные права в зависимости от того является ли он администратором.

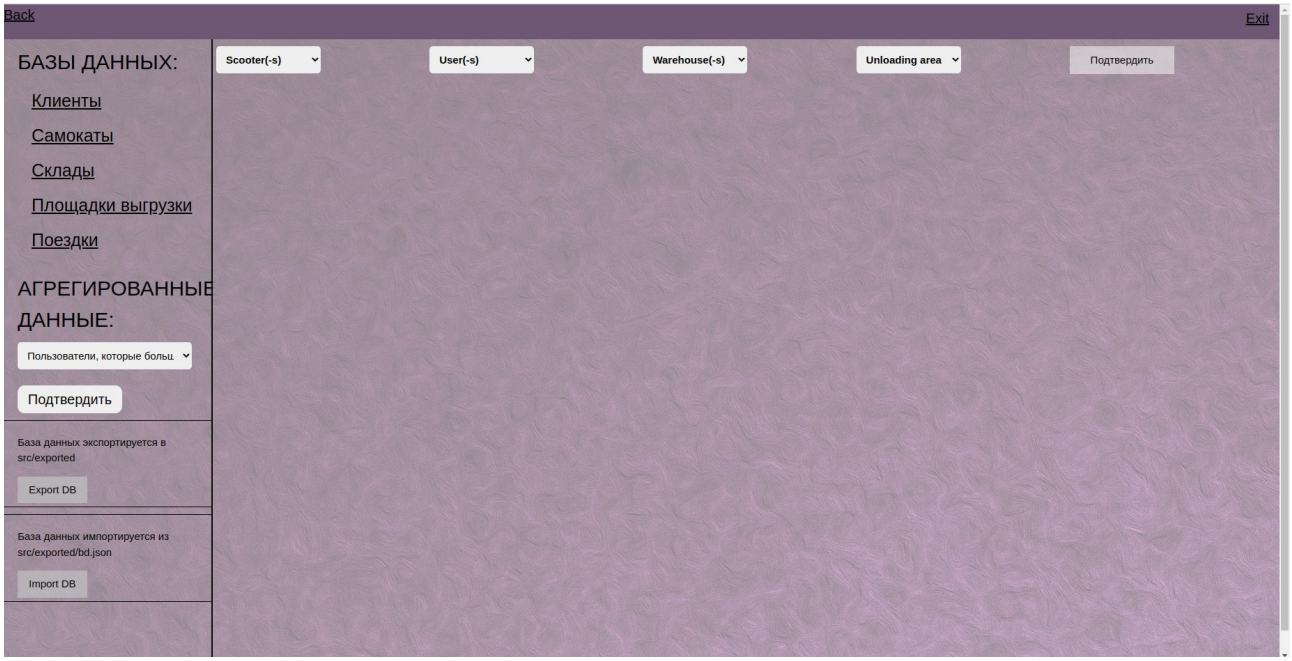
Для рядовых пользователей доступен просмотр и заказ самокатов, просмотр тарифов и правил.

Для администратора доступно то же, что для пользователя, а также просмотр базы данных (самокатов, пользователей, складов, площадок выгрузки и поздок), добавление самокатов и изменение данных о существующих самокатах.

4.2. Схема экранов приложения



Главный экран приложения — просмотр карты



Страница с базами данных. Включает в себя ссылки на страницы с таблицами о пользователях, самокатах, складах, площадках выгрузки и поездках, а также кнопки для импорта и экспорта данных.

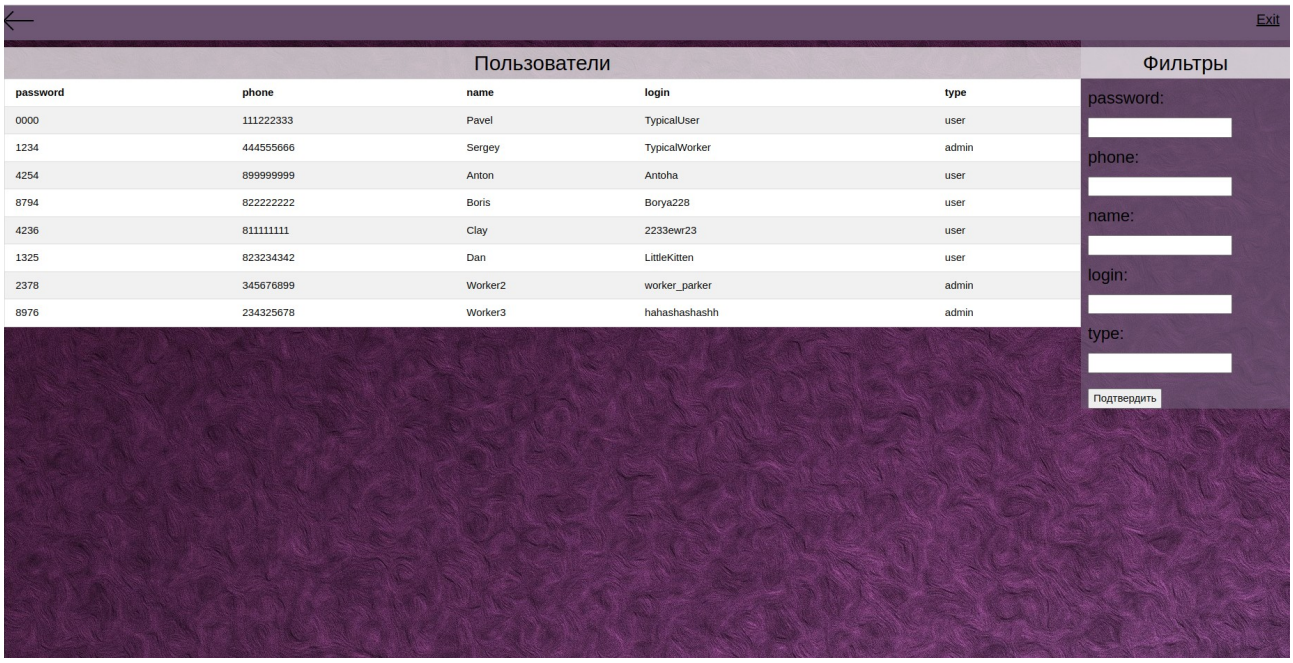
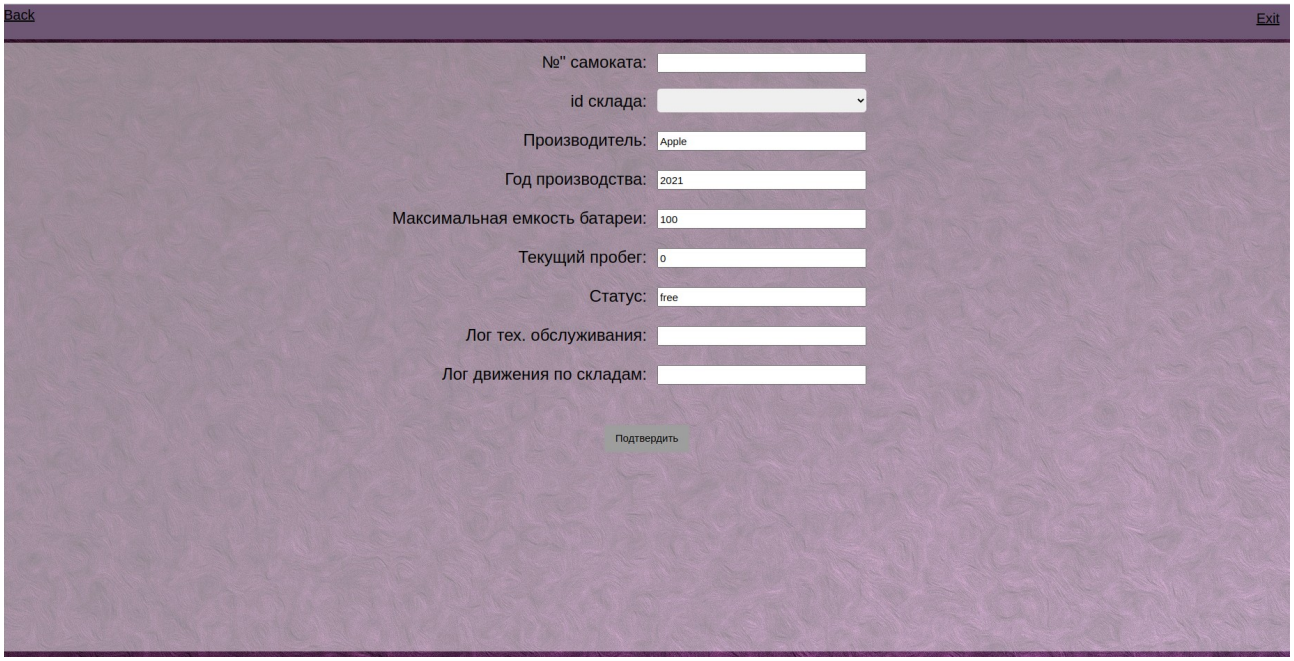


Таблица с данными о пользователях. Поддерживает фильтрацию по полям



Страница добавления и редактирования самокатов

4.3. Использованные технологии

Для хранения данных была использована neo4j, для создания веб-приложения использовался express (node js + pug).

ВЫВОДЫ

Достигнутые результаты:

В ходе работы было разработано веб-приложение для администрирования аренды электросамокатов, в котором возможен просмотр данных администраторами, а также добавление и редактирование информации о самокатах. Для хранения данных и управления ими использовалась СУБД neo4j.

Будущее развитие решения:

Реализация всех сценариев использования.

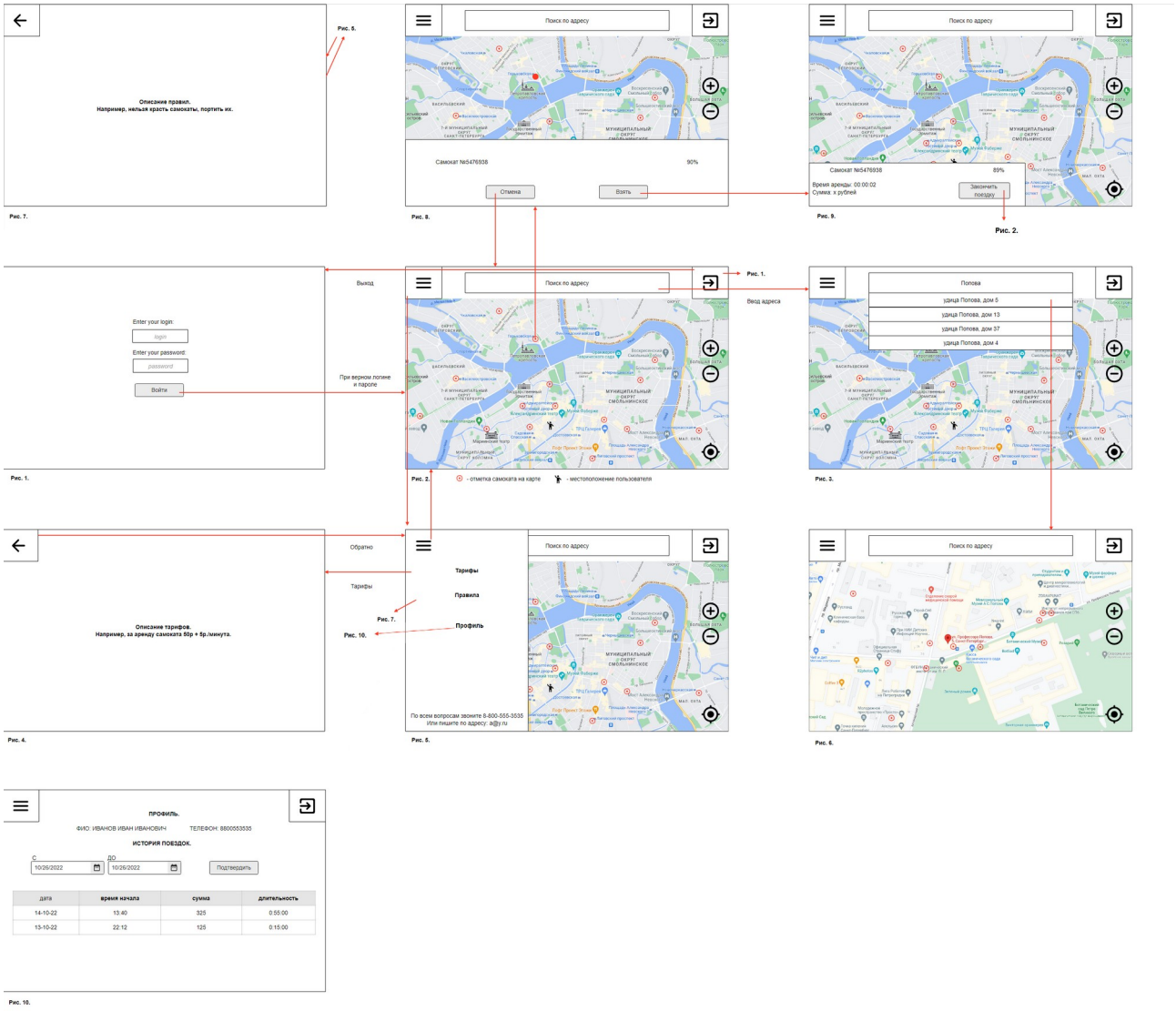
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация neo4j: https://neo4j.com/
2. Документация express: https://expressjs.com/

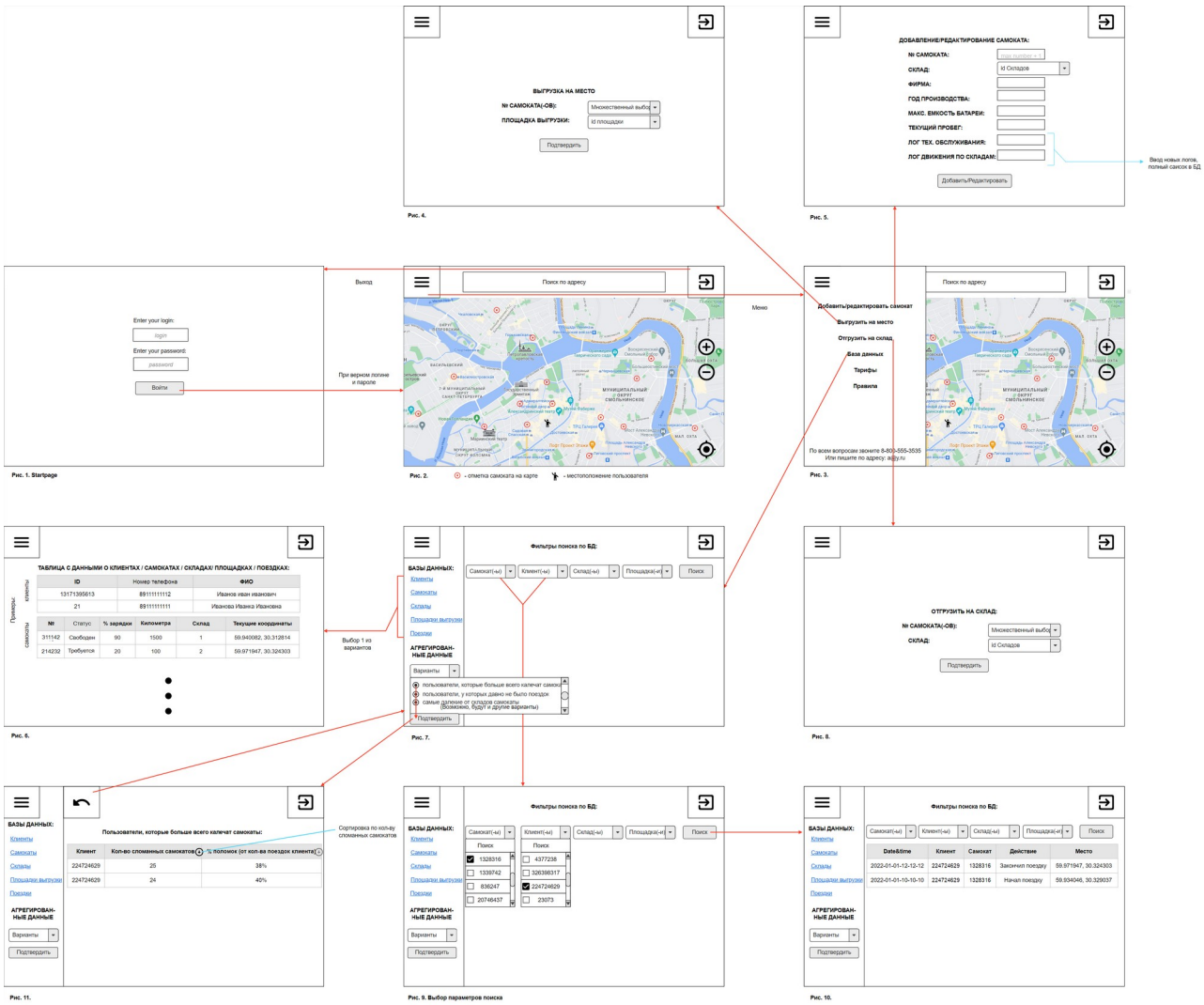
ПРИЛОЖЕНИЕ А

МАКЕТ ДАННЫХ

Макет пользователя:



Макет администратора:


ВЫГРУЗКА НА МЕСТО
№ САМОКАТА(-ОВ):
Множественный выбор
ПЛОЩАДКА ВЫГРУЗКИ:
id площадки
Подтвердить
Рис. 4.
ДОБАВЛЕНИЕ/РЕДАКТИРОВАНИЕ САМОКАТА:
№ САМОКАТА:
СКЛАД:
id Складов
ФИРМА:
ГОД ПРОИЗВОДСТВА:
МАКС. ЕМКОСТЬ БАТАРЕИ:
ТЕКУЩИЙ ПРОБЕГ:
ЛОГ ТЕХ. ОБСЛУЖИВАНИЯ:
ЛОГ ДВИЖЕНИЯ ПО СКЛАДАМ:
Добавить/Редактировать
Ввод новых логов, полный сансок в БД
Рис. 5.
Enter your login:
login
Enter your password:
password
Войти
Рис. 1. Startpage
Выход
При верном логине и пароле
Поиск по адресу
Рис. 2.
- отметка самоката на карте
- местоположение пользователя
Меню
Добавить/редактировать самокат
Выгрузить на место
Отгрузить на склад
База данных
Тарифы
Правила
По всем вопросам звоните 8-800-555-3535
Или пишите по адресу: a@y.ru
Рис. 3.
ТАБЛИЦА С ДАННЫМИ О КЛИЕНТАХ / САМОКАТАХ / СКЛАДАХ/ ПЛОЩАДКАХ / ПОЕЗДКАХ:
Примеры: клиенты
ID
Номер телефона
ФИО
13171395613
89111111112
Иванов иван иванович
21
89111111111
Иванова Иванна Ивановна
самокаты
№
Статус
% зарядки
Километра
Склад
Текущие координаты
311142
Свободен
90
1500
1
59.940082, 30.312814
214232
Требуется
20
100
2
59.971947, 30.324303
Рис. 6.
Выбор 1 из вариантов
ФИЛЬТРЫ ПОИСКА ПО БД:
БАЗЫ ДАННЫХ:
Клиенты
Самокаты
Склады
Площадки выгрузки
Поездки
АГРЕГИРОВАННЫЕ ДАННЫЕ
Варианты
пользователи, которые больше всего налечат самокатов
пользователи, у которых давно не было поездок
самые удалене от складов самокаты
(Возможно, будут и другие варианты)
Подтвердить
Рис. 7.
ОТГРУЗИТЬ НА СКЛАД:
№ САМОКАТА(-ОВ):
Множественный выбор
СКЛАД:
id Складов
Подтвердить
Рис. 8.
Пользователи, которые больше всего налечат самокаты:
Сортировка по кол-ву сломанных самокатов
Клиент
Кол-во сломанных самокатов
% поломок (от кол-ва поездок клиента)
224724629
25
38%
224724629
24
40%
Рис. 11.
Самокат(-ы)
Клиент(-ы)
Склад(-ы)
Площадка(-и)
Поиск
1328316
1339742
836247
20746437
4377238
326398317
224724629
23073
Рис. 9. Выбор параметров поиска
Date&time
Клиент
Самокат
Действие
Место
2022-01-01-12-12-12
224724629
1328316
Закончил поездку
59.971947, 30.324303
2022-01-01-10-10-10
224724629
1328316
Начал поездку
59.934046, 30.329037
Рис. 10.