**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**

**по дисциплине «Введение в нереляционные базы данных»**

**Тема: Веб-приложение для поиска проблем с лесными насаждениями по спутниковым снимкам**

| Студенты гр. 0303 |  | Болкунов В.О. |
| --- | --- | --- |
|  |  | Давыдов М.Д. |
|  |  | Парамонов В.В. |
| Преподаватель |  | Заславский М.М. |

Санкт-Петербург

2023

**ЗАДАНИЕ**

| Студенты  Болкунов В.О.  Давыдов М.Д.  Парамонов В.В. | | |
| --- | --- | --- |
| Группа 0303 | | |
| Тема проекта: Веб-приложение для поиска проблем с лесными насаждениями по спутниковым снимкам. | | |
| Исходные данные:  Необходимо реализовать веб-приложение загрузки, обработки и хранения спутниковых снимков. В качестве базы данных использова MongoDB, а в качестве платформы для развертывания приложения - Docker. | | |
| Содержание пояснительной записки:  «Содержание»  «Введение»  «Сценарии использования»  «Модель данных»  «Разработанное приложение»  «Выводы»  «Приложения»  «Литература» | | |
| Предполагаемый объем пояснительной записки:  Не менее 15 страниц. | | |
| Дата выдачи задания: 01.09.2023 | | |
| Дата сдачи реферата: 20.12.2023 | | |
| Дата защиты реферата: 25.12.2023 | | |
| Студенты |  | Болкунов В.О. |
|  |  | Давыдов М.Д. |
|  |  | Парамонов В.В. |
| Преподаватель |  | Заславский М.М. |

**АННОТАЦИЯ**

В данной работе описан процесс проектирования и разработки системы гео-мониторинга, которая представляет из себя многоуровневое веб-приложение, позволяющее пользователям работать с картами и объектами на них, и в автоматическом режиме осуществляющее поиск проблем с лесными насаждениями. Архитектура данной системы построена вокруг использования нереляционной базы данных.

**СОДЕРЖАНИЕ**

| 1. | Введение | 6 |
| --- | --- | --- |
| 1.1. | Актуальность решаемой проблемы | 6 |
| 1.2. | Постановка задачи | 6 |
| 1.3. | Предлагаемое решение | 6 |
| 1.4. | Качественные требования к решению | 6 |
| 2. | Сценарии использования | 7 |
| 2.1. | Макет UI | 7 |
| 2.2. | Сценарии использования для разных задач | 7 |
| 2.3. | Вывод о работе с базой данных | 8 |
| 3. | Модель данных | 9 |
| 3.1. | Нереляционная модель | 9 |
| 3.2. | Аналог модели для SQL СУБД | 17 |
| 3.3. | Сравнение | 25 |
| 4. | Разработанное приложение | 28 |
| 4.1. | Краткое описание | 28 |
| 4.2. | Использованные технологии | 28 |
| 4.3. | Снимки экрана приложения | 28 |
| 5. | Выводы | 32 |
| 5.1. | Достигнутые результаты | 32 |
| 5.2. | Недостатки и пути для улучшения полученного решения | 32 |
| 5.3. | Будущее развитие решения | 32 |
| 6. | Приложения | 33 |
| 6.1. | Документация по сборке и развертыванию | 33 |
| 6.2. | Инструкция для пользователя | 33 |
| 7. | Литература | 37 |

1. **ВВЕДЕНИЕ**

**1.1. Актуальность решаемой проблемы**

В современном мире сложно представить себе жизнь без элементов, созданных из дерева: мебель, бумага, строительные материалы. Леса вырубают в огромных количествах, что несомненно необходимо отслеживать и контролировать. Современные технологии позволяют получить геопривязанные спутниковые снимки, с которыми можно работать для выявления проблем в отслеживаемых лесах.

**1.2. Постановка задачи**

Требуется реализовать систему по обработке спутниковых геопривязанных изображений, с целью поиска проблемных зон в лесных насаждениях и работы с найденными или добавленными объедками на картах.

**1.3. Предлагаемое решение**

Предлагается реализовать веб-приложение с возможностью загрузки в него геопривязанных изображений, обработкой данных изображений алгоритмическими методами и сохранением результата в нереляционной базе данных.

**1.4. Качественные требования к решению**

* Загрузка и обработка .tiff файлов.
* Автоматическое нахождение объектов леса и аномалий.
* Возможность добавлять, редактировать и удалять объекты на картах в ручном режиме.
* Наличие экспорта и импорта объектов в формате json.

1. **СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ**

**2.1. Макет UI**

см. файл “Макет системы.png”

**2.2. Сценарии использования для разных ролей пользователей**

## Пользователь (неавторизованный)

1. Неавторизованный пользователь имеет возможность просматривать глобальную карту, загруженные карты, и найденные объекты в системе.
2. На карте пользователь может выбрать зону поиска объектов, просмотреть их и экспортировать в формате json.
3. Неавторизованный пользователь может войти в систему.

## Пользователь (авторизованный)

1. Авторизованному пользователю доступны все возможности неавторизованного и дополнительно ему доступны:

* загрузка карт;
* импорт объектов в формате json;
* добавление (разметка) объектов на карте;
* удаление объектов;
* редактирование объектов.

1. Также авторизованный пользователь может просмотреть и отредактировать свой профиль (логин, пароль, имя); и выйти из профиля.
2. При работе с картой доступны следующие режимы:

* Добавление объекта

В данном режиме пользователь может разметить область на карте соответствующую некоторому объекту, дать ему название и сохранить его.

* Редактирование аномалии

В режиме редактирования можно выбрать объект и изменить его название или занимаемую область

* Удаление аномалии

При нажатии на объект он выделяется и пользователь может его удалить.

**Администратор:**

1. Администратор может выгрузить и загрузить дамп всей базы данных системы.

**2.3. Вывод о работе с базой данных**

Исходя из возможностей загрузки карт и редактирования объектов только для авторизованных пользователей, можно сделать вывод что операции чтения будут преобладать, так как потенциально меньшее число пользователей имеют возможность записи изменений, а также потому что карты и объекты на них чаще ищут и просматривают чем загружают.

1. **МОДЕЛЬ ДАННЫХ**

**3.1. Нереляционная модель**

NoSQL (Mongo DB [2]):

**Коллекции и сущности:**

### objects:

Назначение: хранение областей объектов на общей карте.

Типы данных полей:

* \_id: objectId
* type: string - тип объекта
* name: string - имя объекта
* color: string - цвет выделения объекта
* update: {

user\_id: objectId,

datetime: date

}: array - история изменения объекта пользователями

* location: {

type: string

coordinates: array

}: GeoJSON - расположение объекта

### maps.files:

Назначение: хранение загруженных пользователями карт.

Типы данных полей:

* \_id: objectId
* name: string - название карты
* update: {

user\_id: objectId

datetime: date

}: object - данные о загрузке

* location: {

type: string

coordinates: array

}: GeoJSON - расположение карты

* chunkSize: int - размер чанка
* length: int - длина файла в чанках
* uploadDate: date - дата загрузки файла

### tile\_fs.files:

Назначение: хранение нарезанных фрагментов карт для отображения на глобальной карте на клиенте.

* \_id: objectId
* image\_id: objectId
* x: double - индекс нарезанного фрагмента по ширине
* y: double - индекс нарезанного фрагмента по высоте
* z: double - отдаление
* chunkSize: int - размер чанка файла
* length: int - длина тайла в чанках
* uploadDate: date - дата загрузки тайла

### user:

Назначение: хранение данных зарегистрированных пользователей.

* \_id: objectId
* login: string - логин пользователя
* password: string - пароль
* name: string - имя пользователя
* status: string - статус
* role: string - роль

## Оценка удельного объема информации

Средний размер одного документа коллекции (байт):

* user: 116
* objects: 484

## tile\_fs:

## tile\_fs.files: 137

## tile\_fs.chunks: 90143 (средний размер тайла)

## sum = 90280

* maps:
  + maps.files: 187
  + maps.chunks: 9494277 (средний размер геотиф карт, загружаемых в систему)
  + sum = 9494464

Предположительно в среднем каждый пользователь будет создавать:

* 10 карт (1 maps документ + 298 tile\_fs документов).
* 50 объектов (50 objects документов).

Получаем следующую формулу зависимости объема модели от числа зарегистрированных пользователей (пусть n):

## Избыточность модели

В таблице maps.files избыточно поле user\_id (20 байт).

В таблице tile\_fs.files избыточно поле image\_id (20 байт).

В таблице objects избыточны поля user\_id (20 байт).

Получаем формулу "чистого" объема:

Тогда рассчитаем избыточность как отношение объема модели к "чистому" объему:

## Направление роста модели

При добавлении объекта из коллекции maps.files будет создаваться 298 объектов коллекции tile\_fs.files.

При добавлении объекта в других коллекциях (tile\_fs.files, user, objects) никаких дополнительных объектов добавляться не будет.

## Запросы к модели

1. Нахождение всех карт с условием фильтрации (@filtration\_cond) и с условием сортировки (@sort\_cond):

* Текст запроса:

db.maps.files.findOne(@filter\_cond).sort(@sort\_cond)

* Пример входных данных

@filtration\_cond = {name: ‘металлострой’}

@sort\_cond = {name: 1, update: {datetime: 1}}

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных коллекций: 1 коллекция.

1. Найти карты, которые находятся в некоторой области @location:

* Текст запроса:

db.maps.files.find({location: {$near: @location}})

* Пример входных данных

@location = {

type: "Polygon",

coordinates: [ [ [ 0 , 0 ] , [ 3 , 6 ] , [ 6 , 1 ] , [ 0 , 0 ] ] ]

}

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных коллекций: 1 коллекция.

1. Удалить карту по id:

* Текст запроса:

db.maps.files.remove({\_id: id})

db.tile\_fs.files.remove({image\_id: id})

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 2 запроса.
* Количество задействованных коллекций: 2 коллекции.

1. Добавить новые карты (пусть необходимо добавить n карт, с определенным приближением, что дает m тайлов на 1 карту):

* Текст запроса:

db.maps.files.insert({map\_1}, …, {map\_n})

db.tile\_fs.files.insert({tile\_1}, …, {tile\_n})

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 2 запроса.
* Количество задействованных коллекций: 2 коллекции.

1. Найти все объекты с условием фильтрации (@filtration\_cond) и с условием сортировки (@sort\_cond):

* Текст запроса:

db.objects.find(@filtration\_cond).sort(@sort\_cond)

* Пример входных данных

@filtration\_cond = {type: ‘аномалия’}

@sort\_cond = {name: 1}

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных коллекций: 1 коллекции.

1. Найти объекты, которые находятся в некоторой области (@some\_location):

* Текст запроса:

db.objects.find({location: {$near: @some\_location}})

* Пример входных данных

@some\_location = {

type: "Polygon",

coordinates: [ [ [ 0 , 0 ] , [ 3 , 6 ] , [ 6 , 1 ] , [ 0 , 0 ] ] ]

}

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных коллекций: 1 коллекции.

1. Удалить объект по id:

* Текст запроса:

db.objects.remove({\_id: id})

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запроса.
* Количество задействованных коллекций: 1 коллекции.

1. Добавить новые объекты @object\_i за пользователей с id = @user\_id\_i во время @timestamp\_i:

* Текст запроса:

db.objects.insert([

{ ...@object1,

update: [ {user\_id: @user\_id1, timestamp: @timestamp1} ]

},

…

{ ...@objectn,

update: [ {user\_id: @user\_idn, timestamp: @timestampn} ]

}])

* Пример входных данных

@objectn = {

type: ‘anomaly’,

name: ‘пожар’,

color: ‘#FF0000’,

location: {

type: "Polygon",

coordinates: [ [ [ 0 , 0 ] , [ 3 , 6 ] , [ 6 , 1 ] , [ 0 , 0 ] ] ]

}

}

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запроса.
* Количество задействованных коллекций: 1 коллекции.

1. Изменить параметры (список параметров со значениями @set\_values, id пользователя и время обновления @set\_user\_values) объекта (с \_id == id):

* Текст запроса:

db.objects.update(

{\_id: id},

{

...@set\_values,

$push: {update: {...@set\_user\_values}}

}

)

* Пример входных данных

@set\_values = {

color: ‘#FF0000’,

location: {

type: "Polygon",

coordinates: [ [ [ 0 , 0 ] , [ 3 , 6 ] , [ 0 , 0 ] ] ]

}

}

@set\_user\_values = {

user\_id: ObjectId(‘...’),

timestamp: ‘2023-10-24’

}

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запроса.
* Количество задействованных коллекций: 1 коллекции.

1. Найти пользователя по логину (@login):

* Текст запроса:

db.users.find({login: @login})

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных коллекций: 1 коллекция.

1. Добавить нового пользователя:

* Текст запроса:

db.users.insert({...@new\_user})

([перечисление параметров пользователя]);

* Пример входных данных

@new\_user = {

login: ‘dima’

password: ‘1337’,

name: ‘Dmitriy’,

status: ‘blocked’,

role: ‘user’

}

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных коллекций: 1 коллекция.

1. Отредактировать данные пользователя, найденного по id (список параметров со значениями @set\_values):
   * Текст запроса:

db.users.update({\_id: id}, {...@set\_values})

* Пример входных данных

@set\_values = {

name: ‘Ivan’,

}

* + Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
  + Количество задействованных коллекций: 1 коллекция.

**3.2. Аналог на SQL**

SQL (MуSQL):

**Коллекции и сущности:**

### objects:

Назначение: хранение областей объектов на общей карте.

Типы данных полей:

* \_id: int(6)
* type: varchar(20) - тип объекта
* name: varchar(20) - имя объекта
* color: varchar(30) - цвет выделения объекта
* location: geometry - расположение объекта

### maps:

Назначение: хранение загруженных пользователями карт.

Типы данных полей:

* \_id: int(6)
* name: varchar(20) - название карты
* user\_id: int(6)
* datetime: datetime(8) - время загрузки карты
* location: geometry - расположение карты
* map: longblob(4294967296) - сам файл изображения карты

### tiles:

Назначение: хранение нарезанных фрагментов карт для отображения на глобальной карте на клиенте.

* \_id: int(6)
* image\_id: int(6)
* x: double(8) - индекс нарезанного фрагмента по ширине
* y: double(8) - индекс нарезанного фрагмента по высоте
* z: double(8) - отдаление
* tile: longblob(4294967296) - сам файл тайла карты

### user:

Назначение: хранение данных зарегистрированных пользователей.

* \_id: int(6)
* login: varchar(20) - логин пользователя
* password: varchar(30) - пароль
* name: varchar(50) - имя пользователя
* status: varchar(20) - статус
* role: varchar(20) - роль

### user\_objects:

Назначение: хранение данных зарегистрированных пользователей.

* \_id: int(6)
* user\_id: int(6)
* datetime: datetime(8) - время обновления объекта пользователем
* object\_id: int(6)

## Оценка удельного объема информации

Средний размер одного документа коллекции (байт):

* objects: 6 + 20 + 20 + 30 + 1620 (средний размер полигона в байт) = 1696 байт.
* maps: 6 + 20 + 6 + 8 + 84 (средний размер прямоугольника в байт) + 9494277 (средний размер геотиф карт, загружаемых в систему) = 9494401 байт.
* tiles: 6 + 6 + 8 + 8 + 8 + 90143 (средний размер тайла) = 90179
* user: 6 + 20 + 30 + 50 + 20 + 20 = 146
* user\_objects: 6 + 6 + 8 + 6 = 26

Предположительно в среднем каждый пользователь будет создавать:

* 10 карт (1 maps документ + 298 tiles документов).
* 50 объектов (50 objects документов + 50 user\_objects документов).

Получаем следующую формулу зависимости объема модели от числа зарегистрированных пользователей (пусть n - число зарегистрированных пользователей).

## Избыточность модели

В таблице tiles избыточно поле image\_id (6 байт).

В таблице maps избыточно поле user\_id (6 байт).

В таблице user\_objects избыточны поля user\_id (6 байт) и object\_id (6 байт).

Получаем формулу "чистого" объема:

Тогда рассчитаем избыточность как отношение объема модели к "чистому" объему:

## Направление роста модели

При добавлении объекта в таблицу maps будет создаваться 298 объектов таблицы tiles.

При добавлении объекта в таблицу objects будет создаваться 1 объект таблицы user\_objects.

При добавлении объекта в других коллекциях (tiles, user, user\_objects) никаких дополнительных объектов добавляться не будет.

## Запросы к модели

Опишем некоторые особые функции для работы с геоданными MySQL, которые будут использоваться в запросах:

MBRIntersects(geometry\_1, geometry\_2) - возвращает 1 или 0, чтобы указать, пересекаются ли минимальные ограничивающие прямоугольники двух geometry geometry\_1 и geometry\_2.

MBRWithin(geometry\_1, geometry\_2) - Возвращает 1 или 0, чтобы указать, находится ли минимальный ограничивающий прямоугольник geometry\_1 внутри минимального ограничивающего прямоугольника geometry\_2.

1. Нахождение всех карт с условием фильтрации (@filtration\_cond) и с условием сортировки (@sort\_cond):

* Текст запроса:

SELECT \* FROM maps

WHERE @filtration\_cond

ORDER BY @sort\_cond;

* Пример входных данных

@filtration\_cond: name = ‘металлострой’

@sort\_cond: name ASC, update ASC

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных таблиц: 1 таблицы.

1. Найти карты, которые пересекают некоторую область @some\_location:

* Текст запроса:

SELECT \* FROM maps

WHERE MBRIntersects(maps.location, @some\_location) OR MBRWithin(maps.location, @some\_location)

* @location: POLYGON [ [ [ 0 , 0 ] , [ 3 , 6 ] , [ 6 , 1 ] , [ 0 , 0 ] ] ]
* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных таблиц: 1 таблицы.

1. Удалить карту по id:

* Текст запроса:

DELETE FROM maps

WHERE maps.\_id = id;

DELETE FROM tiles

WHERE tiles.image\_id = id;

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 2 запроса.
* Количество задействованных таблиц: 2 таблицы.

1. Добавить новые карты (пусть необходимо добавить n карт, с определенным приближением, что дает m тайлов на 1 карту):

* Текст запроса:

INSERT INTO maps VALUES

([перечисление параметров 1 карты]),

([перечисление параметров 2 карты]),

…,

([перечисление параметров n карты]);

INSERT INTO tiles VALUES

([перечисление параметров 1 тайла 1 карты]),

…

([перечисление параметров m тайла n карты]);

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 2 запроса.
* Количество задействованных таблиц: 2 таблицы.

1. Найти все объекты с условием фильтрации (filtration\_cond) и с условием сортировки (sort\_cond):

* Текст запроса:

SELECT \*

FROM objects INNER JOIN user\_objects

ON objects.\_id = user\_objects.objects\_id

WHERE @filtration\_cond

ORDER BY @sort\_cond;

* Пример входных данных

@filtration\_cond: type = ‘аномалия’

@sort\_cond: name ASC

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных таблиц: 2 таблицы.

1. Найти объекты, которые пересекают некоторую область (@some\_location):

* Текст запроса:

SELECT \*

FROM objects INNER JOIN user\_objects

ON objects.\_id = user\_objects.objects\_id

WHERE MBRIntersects(objects.location, @some\_location) OR MBRWithin(objects.location, @some\_location)

* @some\_location: POLYGON [ [ [ 0 , 0 ] , [ 3 , 6 ] , [ 6 , 1 ] , [ 0 , 0 ] ] ]
* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных таблиц: 2 таблицы.

1. Удалить объект по id:

* Текст запроса:

DELETE FROM objects

WHERE objects.\_id = id;

DELETE FROM user\_objects

WHERE user\_objects.object\_id = id;

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 2 запроса.
* Количество задействованных таблиц: 2 таблицы.

1. Добавить новые объекты (пусть необходимо добавить n объектов):

* Текст запроса:

INSERT INTO objects VALUES

([перечисление параметров 1 объекта]),

([перечисление параметров 2 объекта]),

…,

([перечисление параметров n объекта]);

INSERT INTO user\_objects VALUES

([id пользователя + время вставки 1 объекта]),

([id пользователя + время вставки 2 объекта]),

…,

([id пользователя + время вставки n объекта]);

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 2 запроса.
* Количество задействованных таблиц: 2 таблицы.

1. Изменить параметры (список параметров со значениями @set\_values, id пользователя и время обновления @set\_user\_values) объекта (с \_id == id):

* Текст запроса:

UPDATE objects

SET @set\_values

WHERE objects.\_id = id;

UPDATE user\_objects

SET @set\_user\_values

WHERE user\_objects.objects\_id = id;

* Пример входных данных

@set\_values: color = ‘#FF0000’, location = POLYGON [ [ [ 0 , 0 ] , [ 3 , 6 ] , [ 0 , 0 ] ] ])

@set\_user\_values: user\_id = user\_id: ObjectId(‘...’), timestamp = ‘2023-10-24’)

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 2 запроса.
* Количество задействованных таблиц: 2 таблицы.

1. Найти пользователя по логину (@login):

* Текст запроса:

SELECT \* FROM user

WHERE user.login = @login;

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных таблиц: 1 таблица.

1. Добавить нового пользователя @new\_user:

* Текст запроса:

INSERT INTO objects VALUES

@new\_user;

* Пример входных данных

@new\_user: (‘dima’, ‘1337’, ‘Dmitriy’, ‘blocked’, ‘user’)

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных таблиц: 1 таблица.

1. Отредактировать данные пользователя, найденного по id (список параметров со значениями @set\_values):

* Текст запроса:

UPDATE user

SET @set\_values

WHERE user.\_id = id;

* Пример входных данных

@set\_values: name = ‘Ivan’

* Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД и прочих параметров: 1 запрос.
* Количество задействованных таблиц: 1 таблица.

**3.3. Сравнение моделей**

## Удельный объем информации

Для NoSQL СУБД формула объема:

Для SQL СУБД формула объема:

В среднем записи в NoSQL СУБД будут занимать больше места, чем в SQL СУБД, исходя из соотношения формул объема от числа пользователей.

## Запросы по отдельным юзкейсам:

Таблица 2 - Сравнение запросов для NoSQL и SQL реализации базы данных

| **Запрос** | **NoSQL** | | **SQL** | |
| --- | --- | --- | --- | --- |
| запросы | коллекции | запросы | коллекции |
| Нахождение карт | 1 | 1 | 1 | 1 |
| Нахождение карт в области | 1 | 1 | 1 | 1 |
| Удаление карты | 2 | 2 | 2 | 2 |
| Добавление карт | 2 | 2 | 2 | 2 |
| Найти объекты | 1 | **1** | 1 | 2 |
| Найти объекты в области | 1 | **1** | 1 | 2 |
| Удалить объект | **1** | **1** | 2 | 2 |
| Добавить объекты | **1** | **1** | 2 | 2 |
| Изменение параметров объекта | **1** | **1** | 2 | 2 |
| Найти пользователя | 1 | 1 | 1 | 1 |
| Добавить пользователя | 1 | 1 | 1 | 1 |
| Редактировать пользователя | 1 | 1 | 1 | 1 |

Запросы к модели, с помощью которых реализуются сценарии использования, для NoSQL выполняются лучше или также по количеству запросов и затронутых коллекций, в сравнении с SQL.

Для решаемой задачи приоритетнее выбрать NoSQL СУБД, так как она выигрывает по числу запросов и затронутых коллекций, что определяет скорость её работы. Данный фактор важнее, чем небольшой проигрыш в среднем объеме данных занимаемых данными в модели в сравнении с SQL СУБД.

1. **РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ**

**4.1. Краткое описание**

Разработанное приложение может загружать от пользователей геопривязанные фотографии, обрабатывать их с целью поиска объектов на изображении, хранить и наслаивать на глобальную карту мира. Обработку изображений можно корректировать вручную: редактировать границы объектов, добавлять и удалять объекты.

**4.2. Использованные технологии**

Базы данных: MongoDB, redis

Серверное приложение: python, flask, celery, redis

Сервисы обработки снимков: python, rasterio, redis, celery

Клиентское приложение: TypeScript, vuejs, leafletjs, html, scss, vite

Инфраструктура: Nginx, docker, docker compose

**4.3. Снимки экрана приложения**

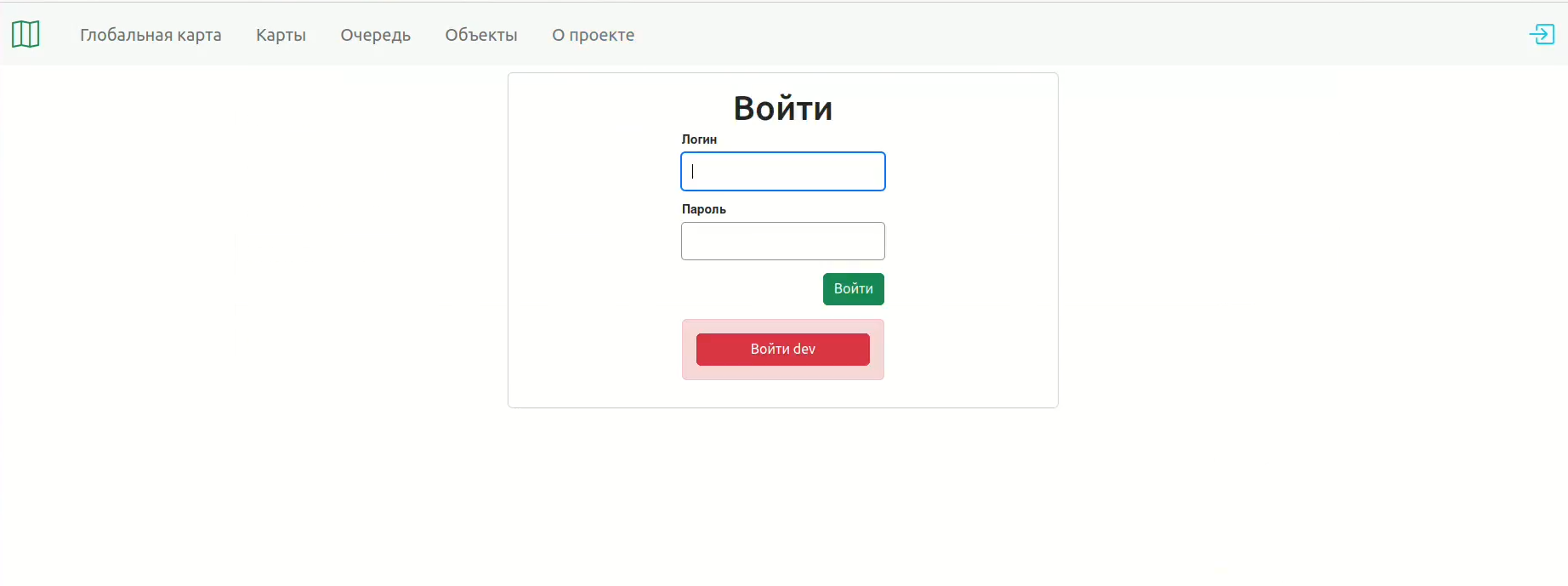


Рисунок 1 - Вход в систему

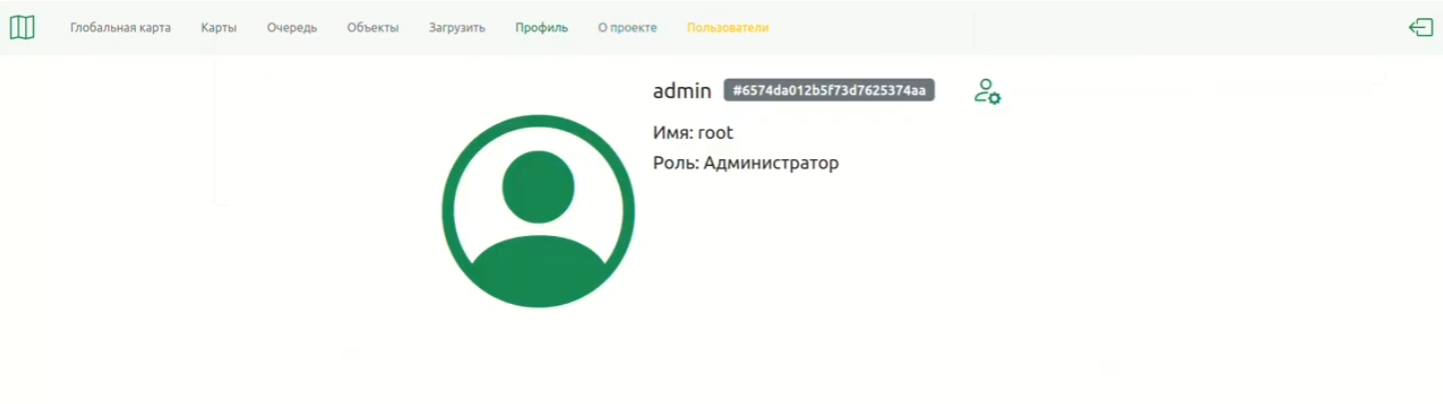


Рисунок 2 - Профиль пользователя

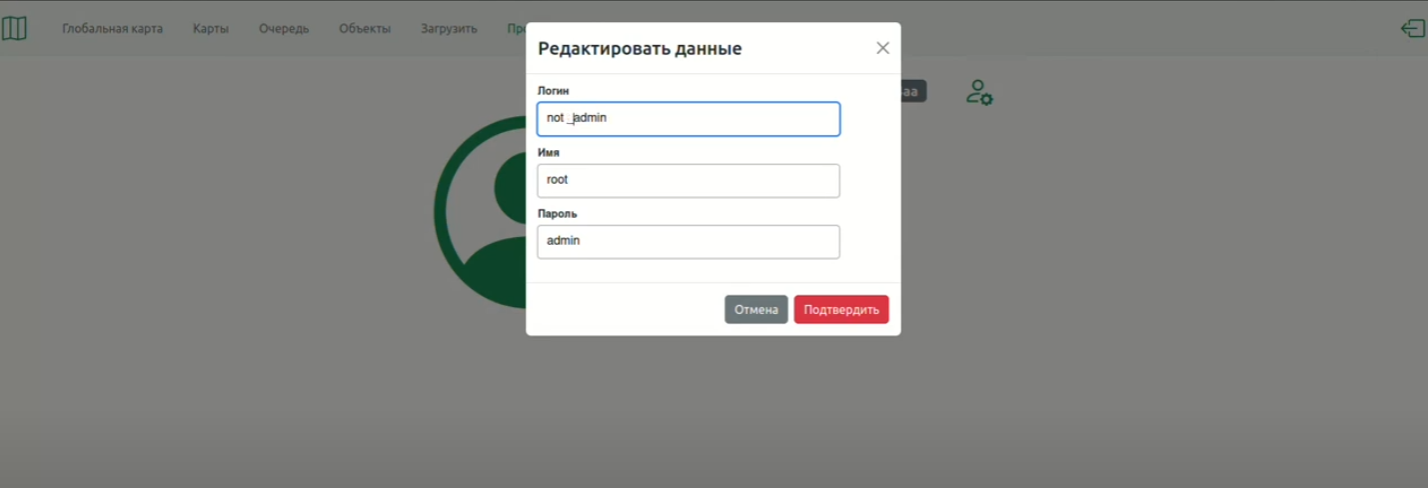


Рисунок 3 - Редактирование профиля

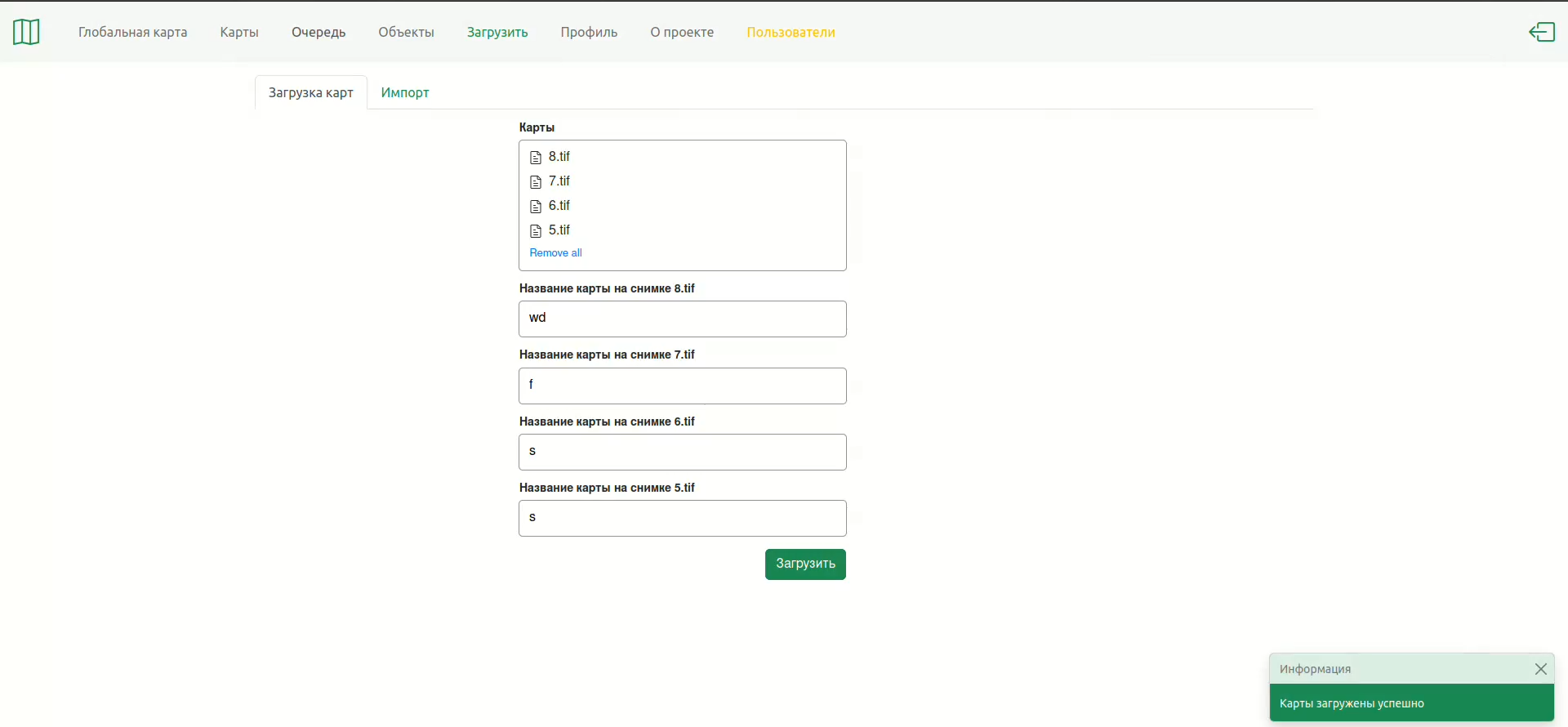


Рисунок 4 - Загрузка карт

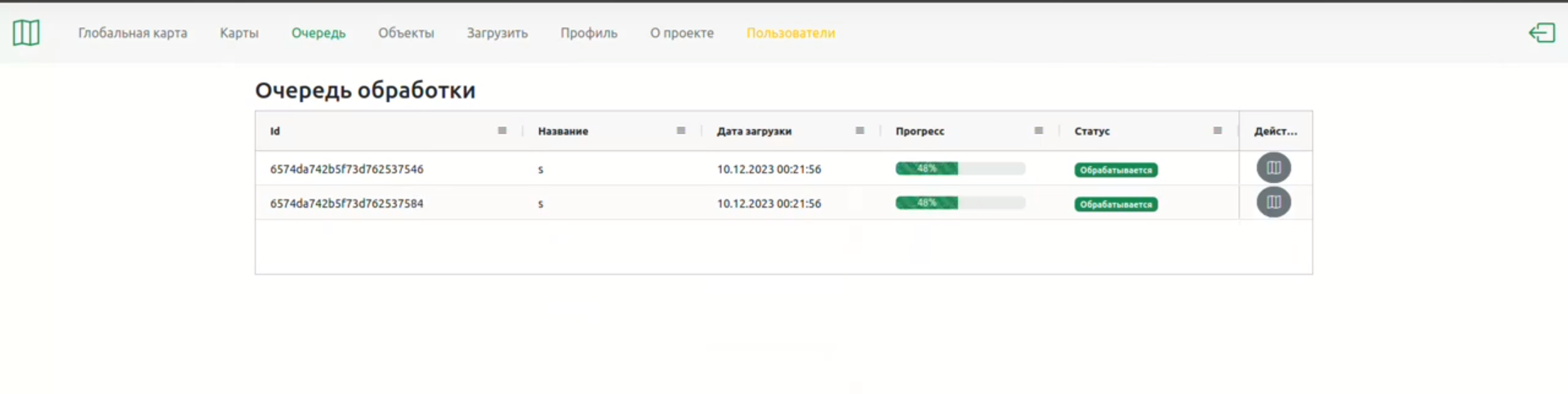


Рисунок 5 - Очередь обработки карт

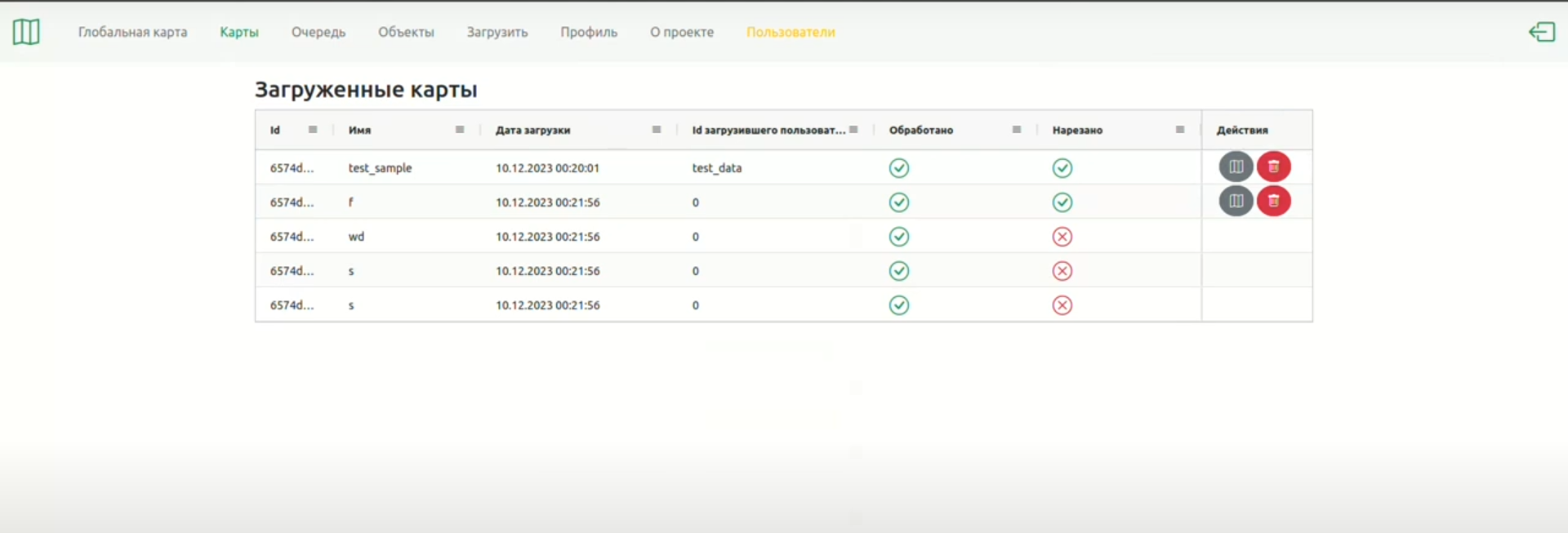


Рисунок 6 - Список карт

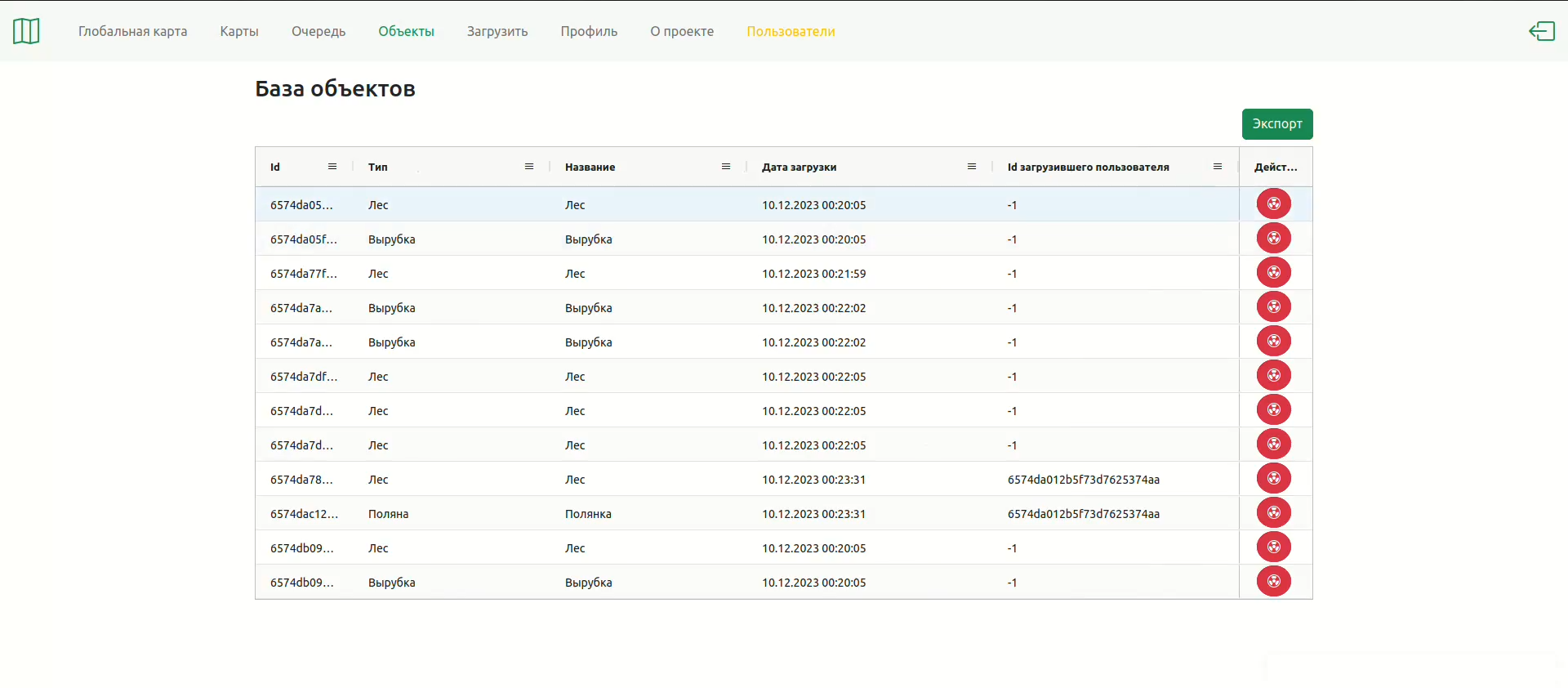


Рисунок 7 - Список объедков в системе

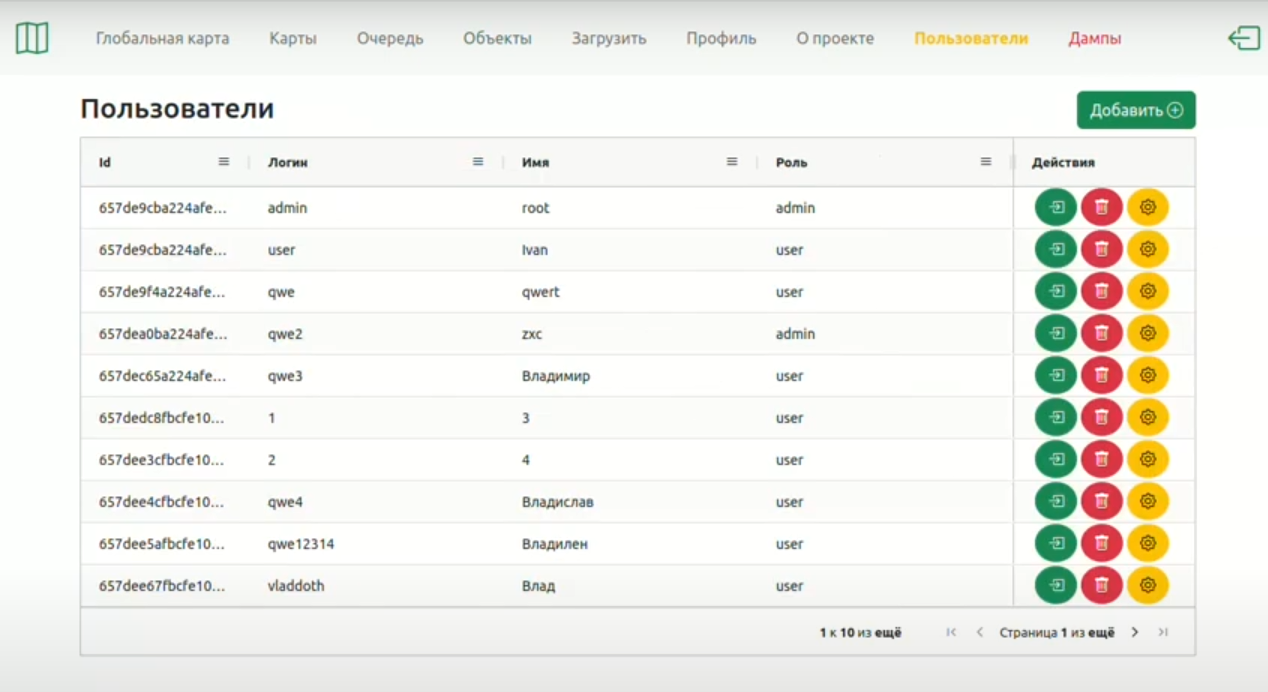


Рисунок 8 - Список пользователей

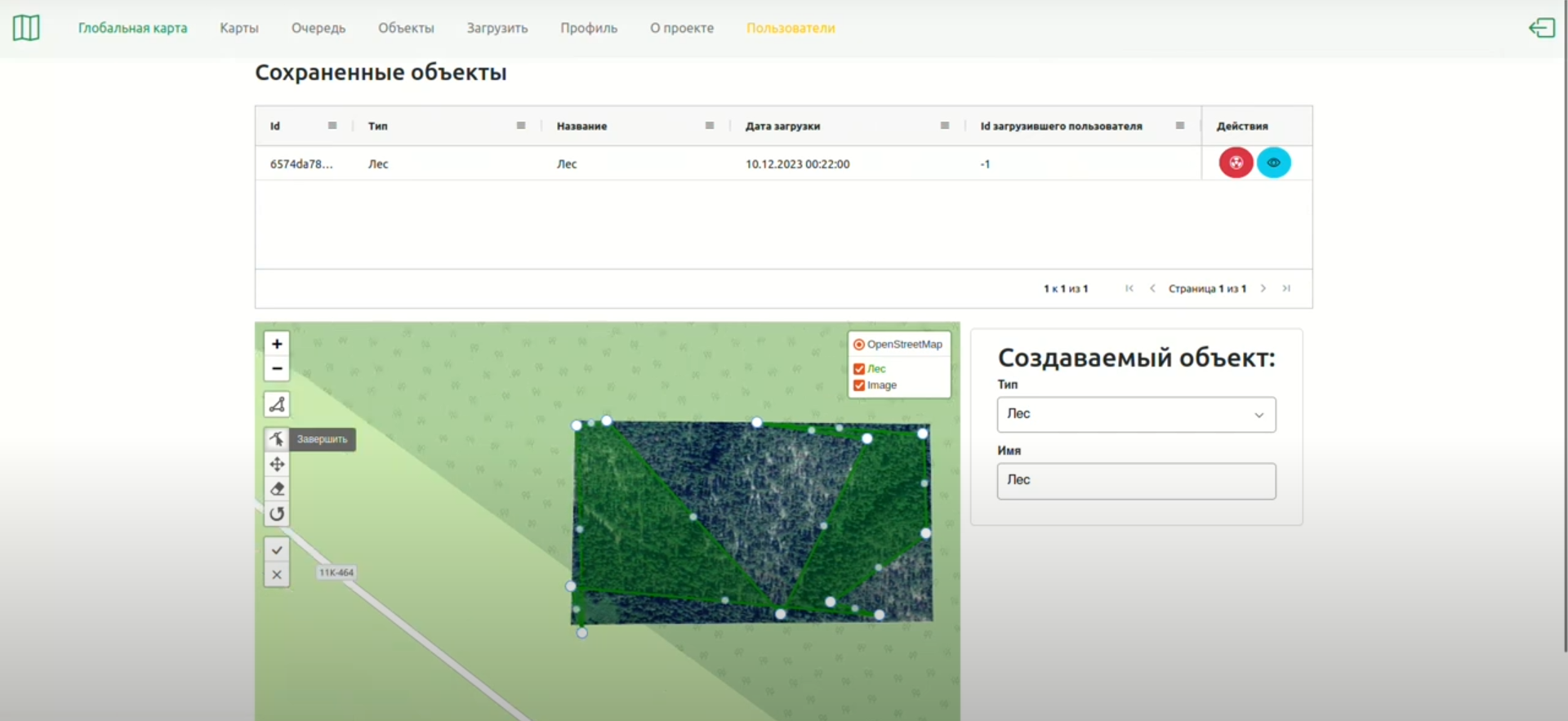


Рисунок 9 - Создание объекта на карте

1. **ВЫВОДЫ**

**5.1. Достигнутые результаты**

Было разработано приложение, способное загружать, хранить и обрабатывать изображения, корректно отображать их все на глобальной карте, а также добавлены возможности изменения найденных объектов, их удаление или добавление пользователем.

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

На данном этапе разработанное приложение неоднозначно определяет объекты на картах: путает дома и лес, вырубки и водоемы. Кроме того, найденные объекты не имеют никаких вычисляемых параметров.

**5.3. Будущее развитие решения**

В качестве вектора развития можно взять увеличение количества находимых объектов, а также увеличить качество их нахождения, например, с помощью нейронных сетей. Кроме того, у найденных объектов можно высчитывать экологические параметры.

1. **ПРИЛОЖЕНИЯ**

**6.1. Документация по сборке и развертыванию**

Для сборки и развертывания необходимо иметь установленный docker и docker compose не ниже 3 версии. После чего достаточно склонировать репозиторий [1] с разработанными приложение и написать команду `docker compose build`. Далее для запуска нужно запустить собранные контейнеры командой `docker compose up`.

**6.2. Инструкция для пользователя:**

**Неавторизованный пользователь:**

* Вход:

1. Перейти на страницу входа с помощью навигационной панели.
2. Ввести логин и пароль.

* Просмотр карт:

1. Перейти на страницу “Карты” с помощью навигационной панели.
2. Найти интересующую карту из списка загруженных карт.
3. Нажать на кнопку в колонке действия, чтобы открыть её просмотр на глобальной карте.

* Просмотр объектов:

1. Перейти на страницу “Объекты” с помощью навигационной панели.
2. Найти интересующую зону из списка созданных объектов.
3. Нажать на кнопку в колонке действия, чтобы открыть детальный просмотр параметров этого объекта.
4. Нажать на кнопку в колонке действия в форме глаза, чтобы открыть просмотр объекта на глобальной карте.

* Экспорт объектов:

1. На вкладке “Глобальная карта” выделить необходимые объекты (они отображаются в таблице сверху глобальной карты).
2. Нажать на кнопку экспорт сверху таблицы с текущими выделенными объектами.

**Авторизованный пользователь:**

* Загрузка карт:

1. Перейти на страницу “Загрузить” с помощью навигационной панели.
2. Выбрать поле загрузки карт.
3. Выбрать загружаемые карты из памяти компьютера.
4. Назвать каждую загружаемую карту.
5. Нажать на кнопку “Загрузить”.

* Импорт объектов в формате json:

1. Перейти на страницу “Загрузить” с помощью навигационной панели.
2. Перейти на вкладку “Импорт”.
3. Выбрать поле загрузки json объектов.
4. Выбрать загружаемый файл json.
5. Нажать на кнопку “Загрузить”.

* Добавление (разметка) объектов на карте:

1. На вкладке “Глобальная карта” нажать на кнопку добавления объектов в меню на карте справа.
2. Нажатиями на карту поставить точки полигона зоны объекта.
3. Выбрать тип и название объекта в полях справа от карты.
4. Нажать на кнопку в виде галочки в меню на карте справа для сохранения добавления объекта.

* Удаление объектов:

1. На вкладке “Глобальная карта” нажать на кнопку удаления объектов в меню на карте справа.
2. Нажать на карте на объект, который необходимо удалить.
3. Нажать на кнопку “Завершить” справа от кнопки удаления.
4. Нажать на кнопку в виде галочки в меню на карте справа для сохранения удаления объекта.

* Редактирование объектов:

1. На вкладке “Глобальная карта” нажать на кнопку редактирования объектов в меню на карте справа.
2. Нажатиями левой кнопкой мыши можно добавлять и перемещать отдельные точки редактируемого объекта. Правая кнопка мыши удаляет точки редактируемого объекта.
3. Нажать на кнопку “Завершить” справа от кнопки редактирования.
4. Нажать на кнопку в виде галочки в меню на карте справа для сохранения удаления объекта.

* Просмотр профиля:

1. Перейти на страницу “Профиль” с помощью навигационной панели.

* Редактирование профиля:

1. Перейти на страницу “Профиль” с помощью навигационной панели.
2. Нажать на кнопку редактирования данных пользователя справа от логина пользователя.
3. Ввести новые данные.
4. Нажать “Подтвердить”.

* Выход из профиля:

1. Нажать на кнопку выхода из аккаунта на навигационной панели.

**Администратор:**

* Создать дамп:

1. Перейти на страницу “Дампы” с помощью навигационной панели.
2. Перейти на вкладку “Экспорт‘.
3. Нажать на кнопку “Загрузить”.

* Загрузить дамп:

1. Перейти на страницу “Дампы” с помощью навигационной панели.
2. Перейти на вкладку “Импорт”.
3. Выбрать поле загрузки json дампа.
4. Выбрать загружаемый файл json.
5. Нажать на кнопку “Загрузить”.

1. **ЛИТЕРАТУРА**

1. Репозиторий проекта // nosql2h23-ecology URL: <https://github.com/moevm/nosql2h23-ecology>

2. Документация базы данных mongo db // What is MongoDB? URL: <https://www.mongodb.com/docs/manual/>