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: Каталог и агрегатор отзывов на озера Ленинградской области

Студентка гр. 1304 _____ Нго Т.Й.

Студентка гр. 1303 _____ Хабибуллина А.М.

Студентка гр. 1303 _____ Хулап О.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2024

ЗАДАНИЕ

Студентка Нго Т.Й.

Студентка Хабибуллина А.М.

Студентка Хулап О.А.

Группа 1303, 1304

Тема: Каталог и агрегатор отзывов на озера Ленинградской области

Исходные данные:

Необходимо реализовать веб-приложение для каталога каталог озер на территории Ленинградской области с использование СУБД Neo4j.

Содержание пояснительной записки:

«Содержание»

«Введение»

«Сценарии использования»

«Модель данных»

«Разработанное приложение»

«Выводы»

«Приложения»

«Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 05.09.2024

Дата сдачи реферата: 24.12.2024

Дата защиты реферата: 24.12.2024

Студентка гр. 1304	_____	Нго Т.Й.
Студентка гр. 1303	_____	Хабибуллина А.М.
Студентка гр. 1303	_____	Хулап О.А.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках ИДЗ разработано веб-приложение, представляющее собой каталог озер на территории Ленинградской области. Приложение включает функционал для просмотра озер, их добавления в списки “Хочу посетить” и “Уже посетил”, возможность оставлять отзывы, поиск по карте, построение маршрутов до озер. Реализована система фильтрации и поиска озер по различным критериям.

Для разработки использованы технологии HTML, CSS, JavaScript, Spring, СУБД Neo4j.

Найти исходный код можно по ссылке: lakes

SUMMARY

As part of the IDZ, a web application was developed, which is a catalog of lakes in the Leningrad Region. The application includes functionality for viewing lakes, adding them to the lists “Want to visit” and “Already visited”, the ability to leave feedback, map search, building routes to lakes. The system of filtering and search of lakes by various criteria is realized.

HTML, CSS, JavaScript, Spring, DBMS Neo4j were used for development.

You can find the source code at the link: lakes

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Актуальность проблемы

Рост интереса к внутреннему туризму и активному отдыху на природе подчеркивает необходимость создания удобных инструментов для поиска и оценки природных объектов. На данный момент многие ресурсы предоставляют лишь ограниченную информацию об озерах Ленинградской области, не предлагая удобных возможностей для фильтрации, поиска и анализа отзывов пользователей. Каталог и агрегатор отзывов на озера позволит объединить данные о природных объектах, облегчить их выбор для отдыха и способствовать популяризации местного туризма.

1.2. Постановка задачи

Задача проекта заключается в разработке веб-приложения, которое позволяет пользователям:

- Находить озера с их описанием и возможностью построения маршрута до них;
- Находить озера по карте;
- Добавлять озера в списки “Хочу посетить” и “Уже посетил”;
- Фильтровать озера по различным параметрам;
- Оставлять отзывы;
- Взаимодействовать с удобным и интуитивным интерфейсом.

Также требуется обеспечить надежное хранение данных и высокую производительность приложения.

1.3. Предлагаемое решение

Для реализации создается веб-приложение с использованием Spring для серверной части и Neo4j для хранения данных. Приложение поддерживает фильтрацию, поиск и публикацию отзывов.

1.4. Качественные требования к решению

Решение должно быть удобным, производительным, надёжным и легко расширяемым.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

2.1. Макет UI

Ниже представлены макеты страниц приложения.

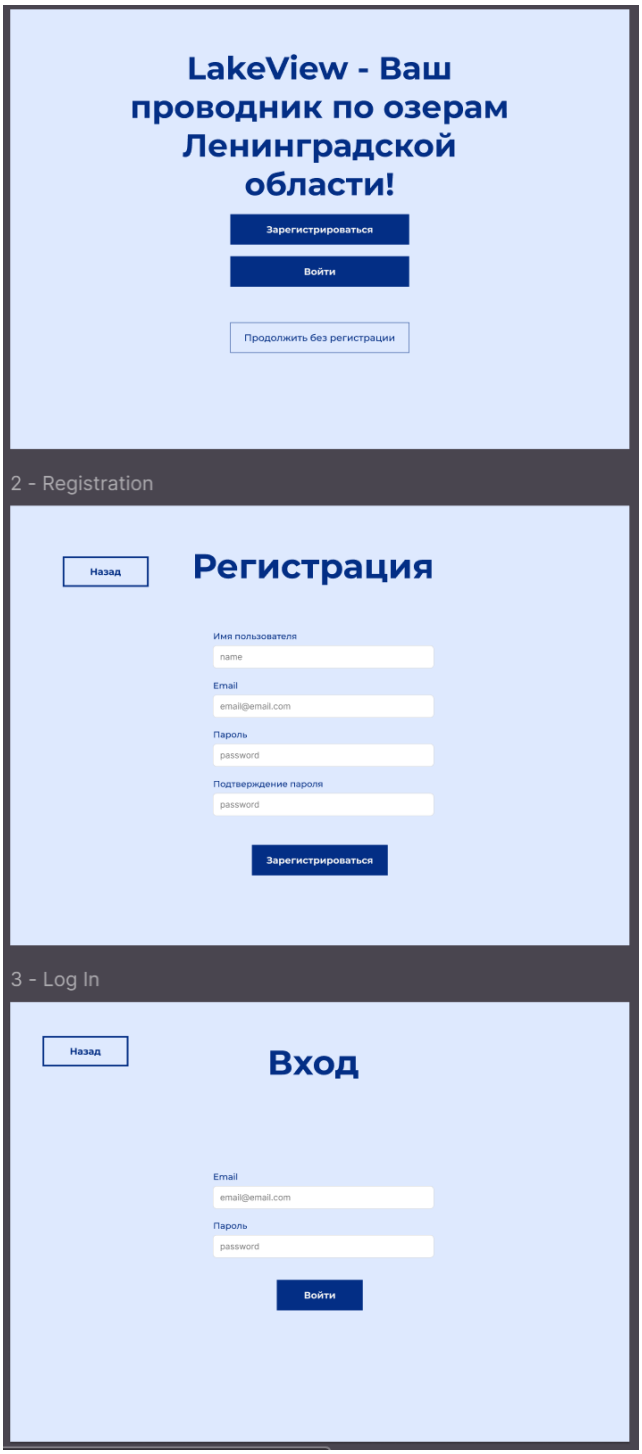


Рисунок 1. Вход/Регистрация

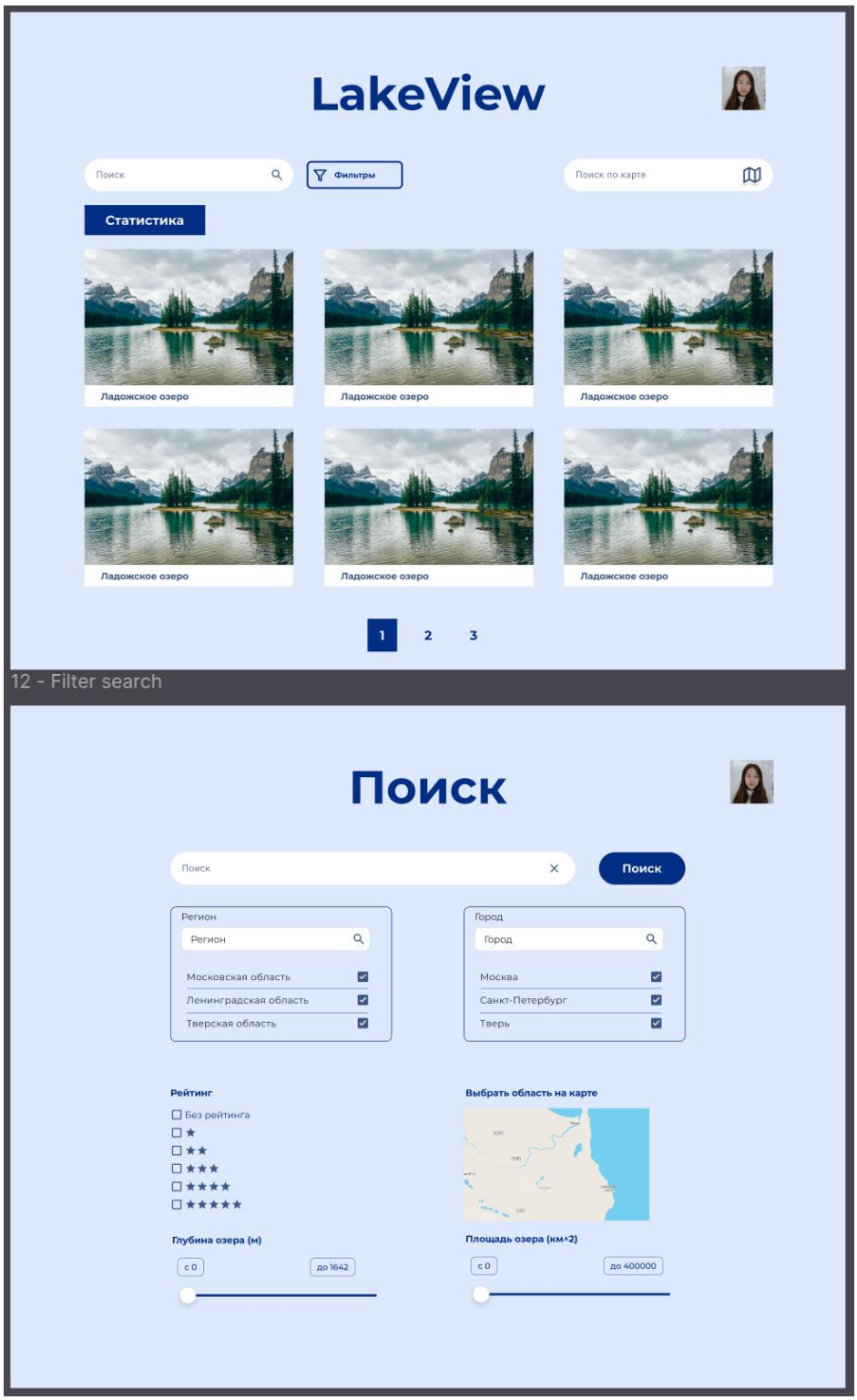


Рисунок 2. Каталог озер и фильтрация

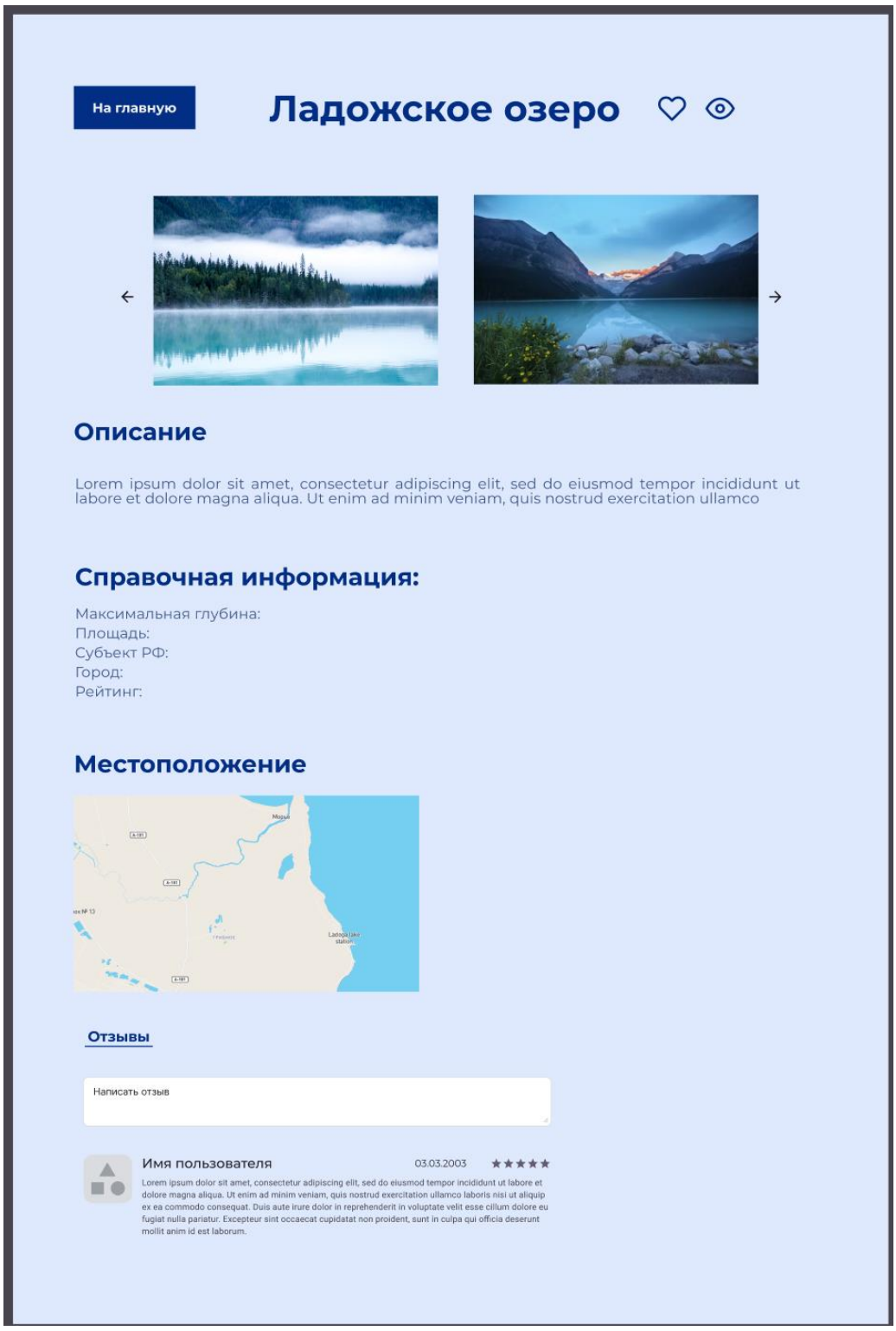


Рисунок 3. Карточка озера

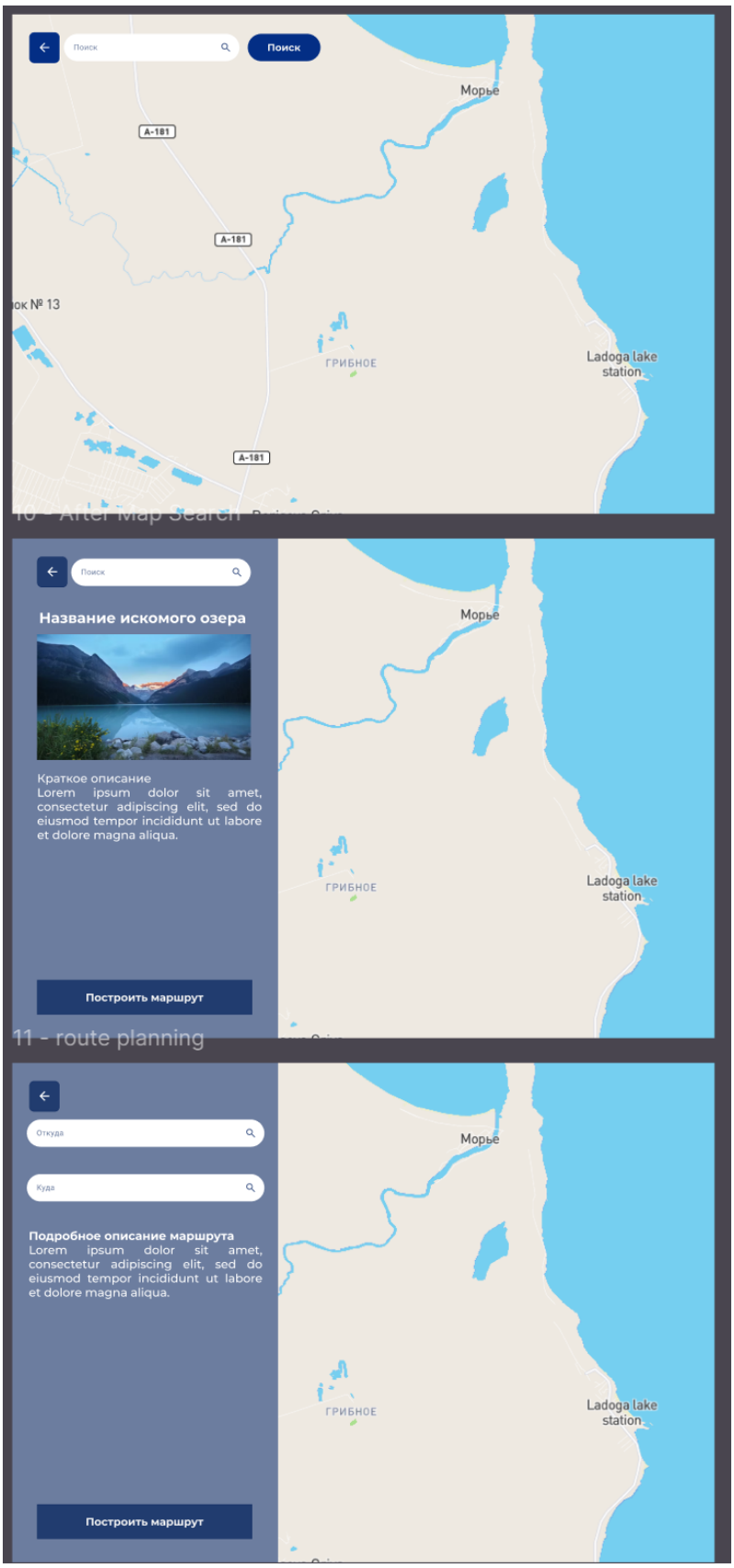


Рисунок 4. Поиск по карте и построение маршрута

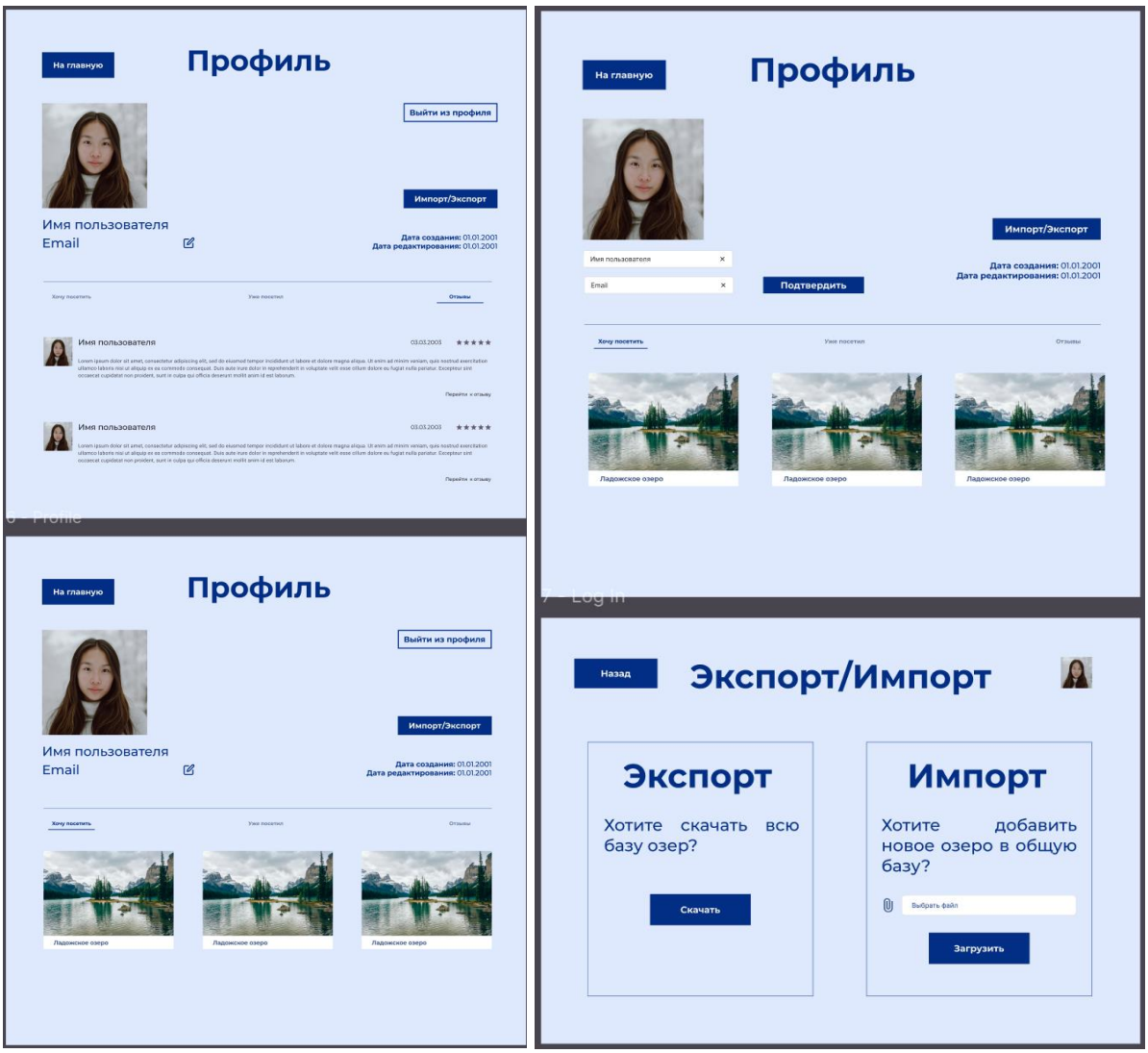


Рисунок 5. Профиль пользователя

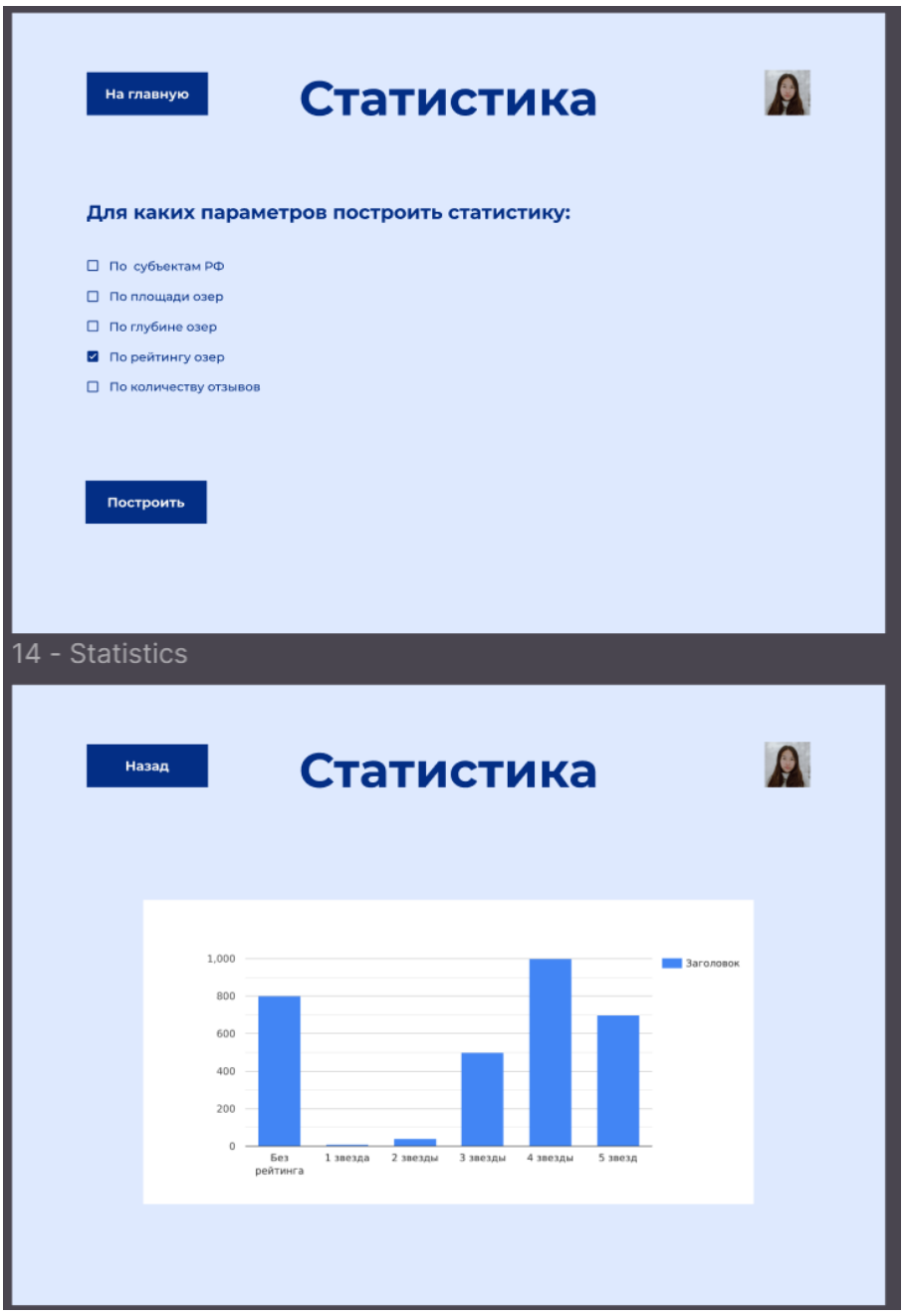
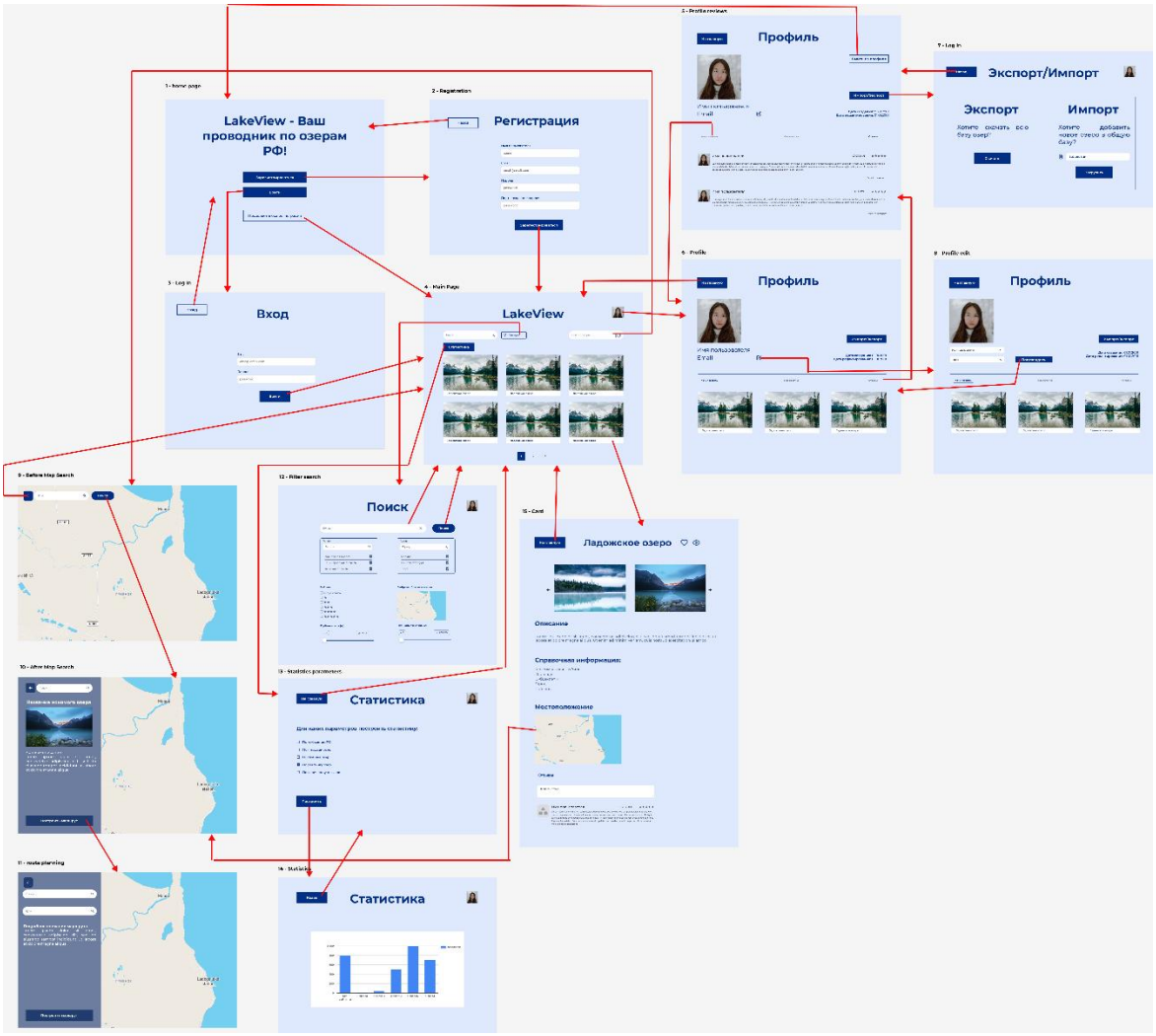


Рисунок 6. Страница построения статистики



2.2. Сценарий использования для импорта данных

а) Сценарий использования: Импорт данных

Цель:

Обновить базу данных.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь переходит на страницу пользователя.
2. Пользователь нажимает кнопку «Импорт/Экспорт».
3. Пользователь попадает на страницу с импортом и экспортом, система отображает кнопку с формой для загрузки файла.
4. Пользователь выбирает файл с расширением .json, из которого хочет считать данные.

5. Пользователь нажимает кнопку "Загрузить".

6. Система проверяет формат загружаемых файлов.

7. Система начинает процесс импорта данных.

8. Система сохраняет данные в базе данных.

Альтернативные сценарии:

7а. Неверный формат файла.

7а.1 Система выводит сообщение об ошибке и предлагает повторить попытку с файлом в правильном формате.

Результат:

База данных обновлена.

б) Сценарий использования: Добавление нового отзыва на озеро

Цель:

Пользователь добавляет отзыв.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь попадает на главную страницу.

2. Пользователь выбирает озеро, к которому хочет добавить отзыв.

3. Открывается страница озера.

4. Пользователь заполняет поле ввода отзыва.

5. Пользователь нажимает кнопку «Отправить».

6. Система сохраняет данные и отображает добавленный отзыв.

Альтернативные сценарии:

6а. Ошибка при сохранении изменений.

6а.1 Система выводит сообщение об ошибке и предлагает повторить попытку.

Результат:

Озеро добавлено в базу данных и отображается в каталоге.

2.3. Сценарий использования для представления данных

Для просмотра данных в виде озер существует два сценария использования: просмотр каталога и просмотр страницы озера.

1. Сценарий использования: Просмотр каталога упражнений

Цель:

Пользователь просматривает каталог озер и применяет фильтры для поиска.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь открывает главную страницу приложения.

2. На экране отображается список озер и кнопка с фильтрами. Изначально, пока пользователь не ввел параметры фильтрации, на странице показываются все озера.

4. При нажатии на кнопку отображаются параметры фильтрации, пользователь отмечает необходимые параметры, нажимает кнопку «Поиск».

5. Система обновляет список озер на основе выбранных фильтров.

6. Пользователь просматривает результаты и выбирает нужное озеро.

Альтернативные сценарии:

4а. Пользователь вводит название озера в поиске и при нажатии на кнопку «Поиск» выполняется запрос.

6а. Пользователь нажимает кнопку «Отмена».

6а.1 Параметры фильтрации очищаются, снова отображаются все озера.

5а. Нет результатов по выбранным фильтрам.

5а.1. Система выводит сообщение, что по текущим критериям ничего не найдено, и предлагает изменить фильтры.

Результат:

Пользователь находит интересующее озеро.

2. Сценарий использования: Просмотр страницы озера

Цель:

Пользователь просматривает детальную информацию об упражнении.

Действующее лицо:

Пользователь

Основной сценарий:

1. Вызывается сценарий «Просмотр каталога озер».

2. Пользователь выбирает озеро в каталоге – нажимает на карточку озера.

3. Система отображает страницу с подробным описанием озера, картой с его местоположением, изображения, а также отзывами на упражнение.

Альтернативные сценарии:

3а. Ошибка при загрузке страницы.

3а.1 Система выводит сообщение об ошибке и предлагает повторить попытку.

Результат:

Пользователь получает полную информацию о выбранном озере.

2.4. Сценарий использования для анализа данных

Сценарий использования: Подсчет статистики в системе

Цель:

Посмотреть статистику данных в системе.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь попадает на главную страницу.

2. Пользователь нажимает кнопку «Статистика».

3. Пользователь выбирает критерий, по которым построить статистику.

4. Система формирует отчет со статистикой (диаграмму).

5. Система отображает отчет пользователю на экране.

Альтернативные сценарии:

4а. Отсутствуют данные для построения статистики.

4а.1 Система выводит сообщение об ошибке и предлагает выбрать другой критерий.

Результат:

Пользователь смотрит статистику, построенную на основе выбранных им данных.

2.5. Сценарий использования для экспорта данных

Сценарий использования: Экспорт данных

Цель:

Экспортировать данные в JSON файл.

Действующее лицо:

Пользователь

Основной сценарий:

1. Пользователь переходит на страницу пользователя.
2. Пользователь нажимает кнопку «Импорт/Экспорт».
3. Пользователь попадает на страницу с импортом и экспортом, система отображает кнопку для экспорта файла.
4. Система формирует JSON файл lakes.json.
5. Файл автоматически загружается на устройство пользователя.

Альтернативные сценарии:

5а. Возникла ошибка при формировании данных.

5а.1 Система выводит сообщение об ошибке и предлагает повторить попытку.

Результат:

Данные экспортированы на устройство пользователя.

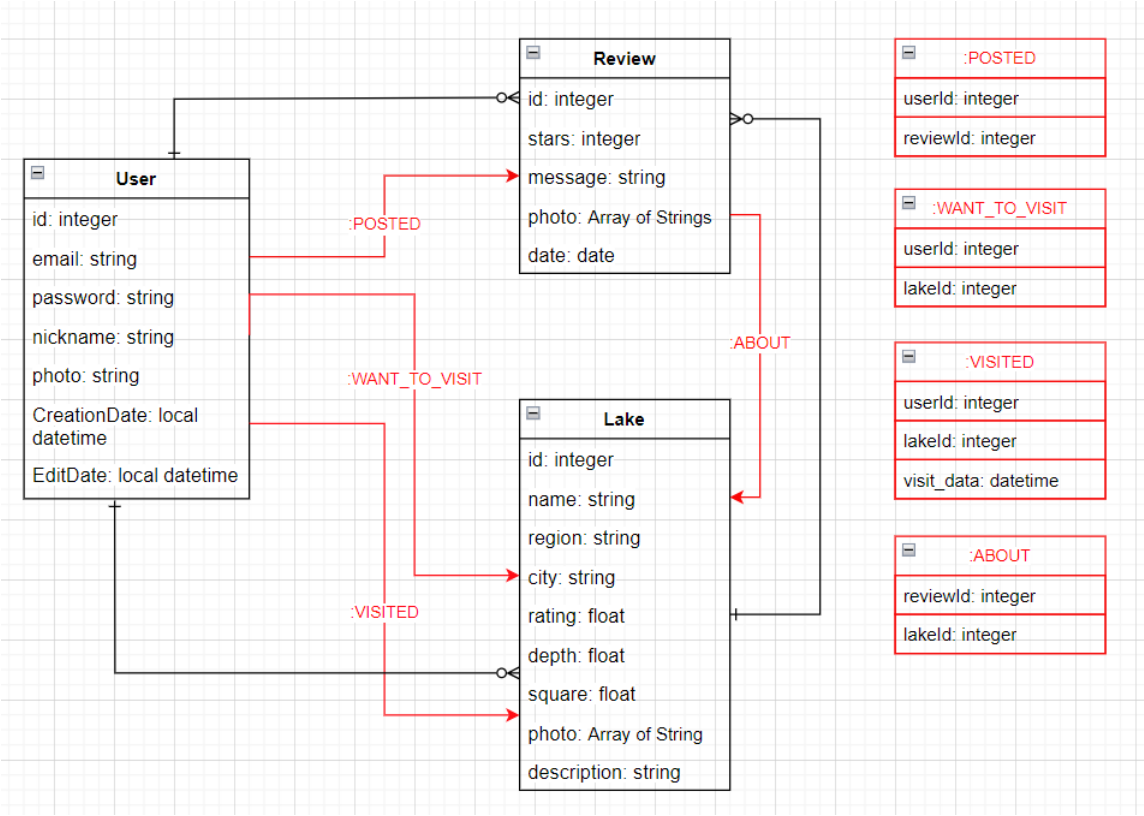
2.6. Вывод

Для нашего решения в равной степени преобладают операции записи и чтения, так как пользователь как активно просматривает данные, так и добавляет, редактирует их в системе.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных

Графическое представление модели



Описание коллекций

Тип	Размер, байт	Описание
INTEGER	4	Целые числа
FLOAT	4	Дробные числа
STRING	500	Строки
LOCAL DATETIME	8	Дата и время без учета часового пояса
DATE	4	Дата без учета часового пояса
BOOL	1	Булева переменная

1. User — модель пользователя. Хранит все личные данные пользователя, имеет связь с Review (:POSTED), данная связь показывает, что пользователь публикует отзывы. Также пользователь имеет связи с Lake. Связь :VISITED показывает, что пользователь может добавить озеро в список "Уже посетил", связь :WANT_TO_VISIT показывает, что пользователь может добавить озеро в список "Хочу посетить".
2. Review — модель отзыва. Хранит информацию об оценке, которую оставил пользователь, сам отзыв и дату его публикации. Имеет связь с Lake (:ABOUT), данная связь указывает, что отзыв написан об определенном озере.
3. Lake — модель озера. Хранит всю информацию об озере и о том, находится ли оно в списках "Хочу посетить" и "Уже посетил".

Оценка объема информации

Оценим удельный объем информации, хранимой в модели.

- User: 4+500×5+8×2=2520 байт
- Lake: 4×4+7×500=3516 байт
- Review: 4×3+500×2=1012 байт

Пусть у каждого пользователя 10 отзывов, всего озер на территории РФ около 3 000 000, тогда при количестве пользователей u:

- $V(u)=(2520+1012\times10)\times u+3516\times3000000=12640\times u+10548000000$

Избыточность данных

Определим избыточные данные в каждой сущности:

- User: избыточный user_id, его размер — 4 байта.
- Lake: избыточный user_id — 4 байта.
- Review: избыточные user_id и lake_id, их общий размер — 8 байт.

Чистый объем данных для каждого объекта:

- User: 2520−4=2516 байт
- Lake: 3516−4=3512 байт
- Review: 1012−8=1004 байта

Формула для чистого объема данных:

- $V_{clean}(u) = (2516 + 1004 \times 10) \times u + 3512 \times 3000000 = 12556 \times u + 10536000000$

Избыточность:

- $V(u)V_{clean}(u) = 12640 \times u + 1054800000012556 \times u + 10548000000 = 1.0067$

Направление роста модели

1. User (Пользователи)

   Каждая новая запись добавляет фиксированный объем (2520 байт), что приводит к линейному росту объема данных.

2. Lake (Озера)

   Рост объема данных линейный, так как каждое новое озеро добавляет фиксированную информацию в базу (3516 байт).

3. Review (Отзывы)

   Каждая новая запись добавляет фиксированный объем (1012 байт), что также ведет к линейному росту объема данных. Также рост может быть квадратичным, если предположить, что каждый пользователь оставляет отзывы о множестве озер.

   Пример данных

   User:

```
CREATE (u:User {
    id: 1,
    email: "example@example.com",
    password: "hashed_password",
    nickname: "User123",
    photo: "user_photo.jpg",
    CreationDate: date("2024-01-01"),
    EditDate: datetime("2024-01-01T12:00:00")
});
```

   User.csv:

```
id,email,password,nickname,photo,CreationDate,EditDate
1,example@example.com,hashed_password,User123,user_photo.jpg,
2024-01-01,2024-01-01T12:00:00
```

   Lake:

```
CREATE (l:Lake {
    id: 1,
    name: "Lake Tahoe",
    region: "California",
    city: "Tahoe City",
    rating: 4.5,
    depth: 501.0,
    square: 122.0,
    photo: ["lake_photo.jpg", "lake_photo2.jpg",
"lake_photo3.jpg"]
    description: "A large, scenic lake in the Sierra Nevada
mountains."
});
```

Lake.csv:

```
id,name,region,city,rating,depth,square,photo,description
1,Lake Tahoe,California,Tahoe
City,4.5,501.0,122.0,["lake_photo.jpg", "lake_photo2.jpg",
"lake_photo3.jpg"],"A large, scenic lake in the Sierra Nevada
mountains."
```

Review:

```
CREATE (r:Review {
    id: 1,
    stars: 5,
    message: "Beautiful lake, highly recommend visiting!",
    photo: ["review_photo.jpg", "review_photo2.jpg"]
    date: date("2024-02-15")
});
```

Review.csv:

```
id,stars,message,photo,date
1,5,"Beautiful lake, highly recommend
visiting!",["review_photo.jpg", "review_photo2.jpg"],2024-02-15
```

POSTED.csv (Связь User -> Review):

```
userId,reviewId
1,1
2,2
```

ABOUT.csv (Связь Review -> Lake):

```
reviewId,lakeId
1,1
2,2
```

VISITED.csv (Связь User -> Lake):

```
userId,lakeId,visit_data
1,1,2024-02-15
2,2,2024-07-20
```

WANT_TO_VISIT.csv (Связь User -> Lake):

```
userId,lakeId
1,2
2,1
```

Примеры запросов

1. **“Регистрация”**

   Проверка, существует ли пользователь с таким email:

```
MATCH (u:User {email: "user@example.com"})
RETURN u;
```

   Создание нового пользователя:

```
CREATE (u:User {
    id: 1,
    email: "user@example.com",
    password: "hashed_password",
    nickname: "username",
    CreationDate: datetime(),
    EditDate: datetime()
});
```

   Количество запросов: 1 запрос на проверку существования пользователя

   + 1 запрос на создание нового пользователя.

   Количество задействованных коллекций: 1

2. **“Вход”**

   Проверка, существует ли пользователь с данным email и паролем:

```
MATCH (u:User {email: "user@example.com", password:
"hashed_password"})
RETURN u;
```

   Количество запросов: 1 запрос для проверки email и пароля.

   Количество задействованных коллекций: 1

3. **“Поиск по списку”**

   Фильтрация озёр по параметрам:

```
MATCH (l:Lake)
WHERE l.region = "Siberia" AND
      l.city = "Irkutsk" AND
      l.rating >= 4.0 AND
      l.depth >= 1000 AND
      l.square >= 5000
RETURN l;
Поиск озера по названию:
MATCH (l:Lake {name: "Baikal"})
RETURN l;
```

   Количество запросов: 1 запрос для фильтрации озёр.

   Количество задействованных коллекций: 1

4. **“Добавление в список ‘Хочу посетить’”**

```
MATCH (u:User {id: 1})-[:WANT_TO_VISIT]->(l:Lake {id: 2})
RETURN u
UNION
MATCH (u:User {id: 1}), (l:Lake {id: 2})
CREATE (u)-[:WANT_TO_VISIT]->(l);
```
Количество запросов: 2

Количество задействованных коллекций: 2

5. **“Добавление в список ‘Уже посетил’”**

```
MATCH (u:User {id: 1})-[:VISITED]->(l:Lake {id: 2})
RETURN u
UNION
MATCH (u:User {id: 1}), (l:Lake {id: 2})
CREATE (u)-[:VISITED]->(l);
```
Количество запросов: 2

Количество задействованных коллекций: 2

6. **“Статистика”**

Статистика по регионам:

```
MATCH (l:Lake)
RETURN l.region, COUNT(l) AS lakes_per_region
ORDER BY lakes_per_region DESC;
```
Количество запросов: 1

Количество задействованных коллекций: 1

Статистика по площади:

```
MATCH (l:Lake)
WITH CASE
    WHEN l.square < 100 THEN 'Малые озёра'
    WHEN l.square >= 100 AND l.square < 1000 THEN 'Средние озёра'
    ELSE 'Крупные озёра'
END AS area_category, COUNT(l) AS lakes_count
RETURN area_category, lakes_count
ORDER BY lakes_count DESC;
```
Количество запросов: 1

Количество задействованных коллекций: 1

Статистика по глубине:

```
MATCH (l:Lake)
WITH CASE
    WHEN l.depth < 50 THEN 'Мелкие озёра'
    WHEN l.depth >= 50 AND l.depth < 200 THEN 'Средние озёра'
    ELSE 'Глубокие озёра'
END AS depth_category, COUNT(l) AS lakes_count
RETURN depth_category, lakes_count
ORDER BY lakes_count DESC;
```

Количество запросов: 1

Количество задействованных коллекций: 1

Статистика по количеству отзывов:

```
MATCH (l:Lake)-[:HAS_REVIEW]->(r:Review)
RETURN l.name AS lake_name, COUNT(r) AS reviews_count
ORDER BY reviews_count DESC;
```

Количество запросов: 1

Количество задействованных коллекций: 2

Статистика по рейтингу:

```
MATCH (l:Lake)
RETURN l.name AS lake_name, l.rating AS lake_rating
ORDER BY l.rating DESC;
```

Количество запросов: 1

Количество задействованных коллекций: 1

7. **Поиск всех озер без посещений и без отзывов**

```
MATCH (l:Lake)
WHERE NOT (l)<-[:VISITED]-(:User) AND NOT (l)<-[:ABOUT]-(:Review)
RETURN l;
```

Количество запросов: 1

Количество задействованных коллекций: 3

8. **Поиск пользователей, которые хотя бы одно озеро посетили, но отзыв не оставили**

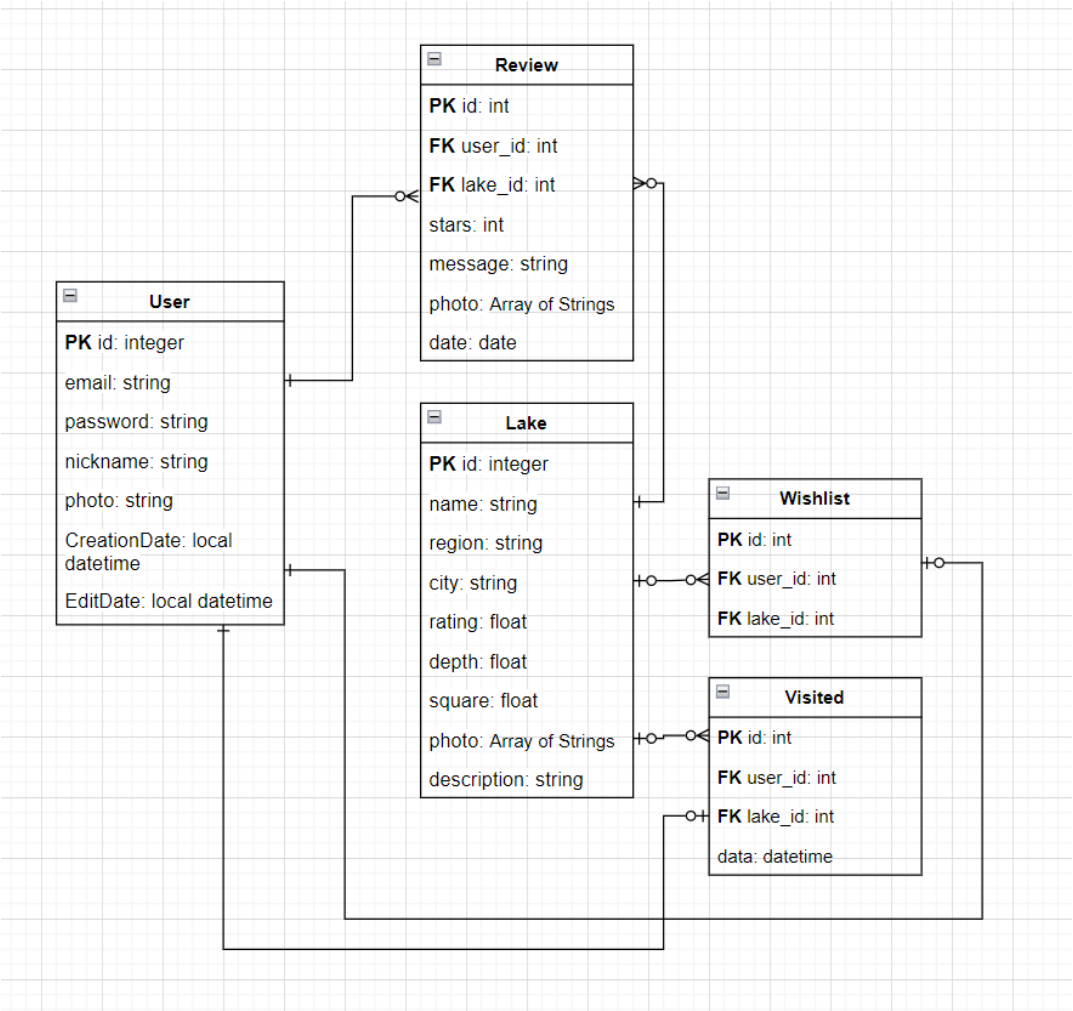
```
MATCH (u:User)-[:VISITED]->(l:Lake)
WHERE NOT (u)-[:POSTED]->(:Review)-[:ABOUT]->(l)
RETURN DISTINCT u;
```

Количество запросов: 1

Количество задействованных коллекций: 3

3.2. Аналог модели данных для SQL СУБД

Графическое представление модели



Описание коллекций n – максимально необходимый размер строки для текста. Оптимальное n = 500 символов.

User - информация о пользователе

Типы данных: - id: integer (4) - email: string (n+1) - password: string (n+1) - nickname: string (n+1) - photo: string (n+1) - CreationDate: local datetime (4) - EditDate: local datetime (4)

Lake - информация об озере

Типы данных: - id: integer (4) - name: string (n+1) - region: string (n + 1) - city: string (n + 1) - rating: float (4) - depth: float (4) - square: float (4) - photo: string (n+1) - description: text (n + 1)

Review - информация об отзыве

Типы данных: - id: integer (4) - user_id: integer (4) - lake_id: integer (4) - stars: integer (4) - message: text (n+1) - photo: string (n+1) - date: local datetime (4)

Wishlist - озера, которые хочет посетить пользователь

Типы данных: - id: integer (уникальный идентификатор записи) - user_id: integer (идентификатор пользователя) - lake_id: integer (идентификатор озера)

Visited - озера, которые уже посетил пользователь

Типы данных: - id: integer (4) - user_id: integer (4) - lake_id: integer (4)

Оценка объема информации

Оценим удельный объем информации, хранимой в модели.

Средний размер одного документа коллекции:

- **User**:

  $$4 \times 3 + (n + 1) \times 4 + 24\,(\text{оверхед строки}) = 2040$$

  байт
- **Review**:

  $$4 \times 5 + 2 \times (n + 1) + 24 = 1046$$

  байт
- **Wishlist**:

  $$4 \times 3 = 12$$

  байт
- **Visited**:

  $$4 \times 3 = 12$$

  байт

Пусть у каждого пользователя 1 Wishlist, 1 Visited, и 10 отзывов. Всего озер на территории РФ около 3 000 000.

- **Lake**:

$$4 \times 4 + 24\,(\text{оверхед строки}) + (n + 1) \times 5 = 2545$$

байт

Объем информации для пользователей:

$$V(u) = u \times (12 + 12 + 2040 + 10 \times 1046) = 12524 \times u + 7635000000$$

Определим избыточные данные в каждой сущности:

- **Users**: избыточное поле — id (4 байта).
- **Reviews**: избыточные поля — user_id (4 байта), lake_id (4 байта).
- **Lakes**: нет явных избыточных полей.
- **Wishlist**: избыточные поля — user_id (4 байта), lake_id (4 байта).
- **Visited**: избыточные поля — user_id (4 байта), lake_id (4 байта).

Чистый объем данных без избыточных полей:

- **Users (пользователи)**:

$$V_{\text{users}} = 2040 - 4 = 2036$$

байт

- **Reviews (отзывы)**:

V_reviews = 1046 - 8 = 1038

байт

- **Lakes (озера)**:

$$V_{\text{lakes}} = 2545$$

байт

- **Wishlist (избранное)**:

$$V_{\text{wishlist}} = 12 - 8 = 4$$

байта

- **Visited (посещенные)**:

$$V_{\text{visited}} = 12 - 8 = 4$$

байта

Формула для чистого объема данных:

$$V_{\text{clean}}(u) = u \times (2 \times 8 + 2036 + 10 \times 1038) = 12432 \times u + 7635000000$$

Избыточность данных

$$\frac{V(u)}{V_{\text{clean}}(u)} = \frac{12524 \times u + 7635000000}{12432 \times u + 7635000000} = 1.0074$$

Направление роста модели

1. **Пользователи (Users)**: Каждая новая запись добавляет фиксированный объем (2040 байт), что приводит к линейному росту объема данных.
2. **Отзывы (Reviews)**: Каждая новая запись добавляет фиксированный объем (1046 байт), что также ведет к линейному росту объема данных. Также рост может быть квадратичным, если предположить, что каждый пользователь оставляет отзывы о множестве озер.
3. **Озера (Lakes)**: Линейный рост. Каждое новое озеро добавляет фиксированный объем данных в базу (около 2545 байт на озеро).

Примеры данных

User.json

```
{
  "users": [
    {
      "id": 1,
      "email": "user1@example.com",
      "nickname": "UserOne",
      "password": "hashed_password",
      "photo": "user_photo.jpg",
      "CreationDate": "2023-01-01T00:00:00Z",
      "EditDate": "2023-01-01T00:00:00Z"
    }
  ]
}
```

Lakes.json

```
{
  "lakes": [
    {
      "id": 1,
```

```
      "name": "Lake One",
      "region": "Region A",
      "city": "City A",
      "rating": 4.5,
      "depth": 20.0,
      "square": 150.0,
      "photo": ["lake1.jpg", "lake2.jpg", "lake3.jpg"],
      "description": "Beautiful lake"
    }
  ]
}
```

Review.json

```
{
  "reviews": [
    {
      "id": 1,
      "user_id": 1,
      "lake_id": 1,
      "stars": 5,
      "message": "Amazing experience!",
      "photo": ["review1.jpg", "review2.jpg", "review3.jpg"],
      "date": "2023-01-02T00:00:00Z"
    }
  ]
}
```

Wishlist.json

```
{
  "wishlist": [
    {
      "id": 1,
      "user_id": 1,
      "lake_id": 2
    }
  ]
}
```

Visited.json

```
{
  "visited": [
    {
      "id": 1,
      "user_id": 1,
      "lake_id": 1,
      "data": "2024-02-15"
    }
  ]
}
```

Примеры запросов

Регистрация

```
SELECT * FROM users WHERE email = 'user@example.com';
INSERT INTO users (email, username, password)
VALUES ('user@example.com', 'username', 'hashed_password');
```

Количество запросов: 2
Количество задействованных коллекций: 1

Вход

```
SELECT * FROM users
WHERE email = 'user@example.com' AND password = 'hashed_password';
```

Количество запросов: 1
Количество задействованных коллекций: 1

Запрос для поиска озёр по критериям

```
SELECT * FROM lakes
WHERE
    (name LIKE '%lake_name%' OR lake_name IS NULL) AND
    (region = 'region_name' OR region IS NULL) AND
    (city = 'city_name' OR city IS NULL) AND
    (rating >= 3 OR rating IS NULL) AND
    (area >= 10 OR area IS NULL) AND
    (depth >= 5 OR depth IS NULL);
```

Количество запросов: 1
Количество задействованных коллекций: 1

Запрос для добавления озера в список «Хочу посетить»

```
SELECT COUNT(*)
FROM wishlist
WHERE user_id = 123 AND lake_id = 456;

# Если озеро не добавлено, добавить его в список
INSERT INTO wishlist (user_id, lake_id)
VALUES (123, 456);
```

Количество запросов: 2
Количество задействованных коллекций: 1

Запрос для добавления озера в список «Уже посетил»

```
SELECT COUNT(*)
FROM visited
WHERE user_id = 123 AND lake_id = 456;
```

Если озеро не добавлено, добавить его в список

```
INSERT INTO visited (user_id, lake_id)
VALUES (123, 456);
```

Количество запросов: 2
Количество задействованных коллекций: 1

Запрос для экспорта данных озёр

```
SELECT * FROM lakes;
```

Количество запросов: 1
Количество задействованных коллекций: 1

Запрос для импорта данных (например, после загрузки файла)

```
LOAD DATA INFILE 'file_path'
INTO TABLE lakes
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Количество запросов: 1
Количество задействованных коллекций: 1

Статистика по субъектам РФ

```
SELECT region, COUNT(*) AS lake_count
FROM lakes
GROUP BY region;
```

Количество запросов: 1
Количество задействованных коллекций: 1

Статистика по площади озёр

```
SELECT area, COUNT(*) AS lake_count
FROM lakes
GROUP BY area;
```

Количество запросов: 1
Количество задействованных коллекций: 1

Статистика по глубине озёр

```
SELECT depth, COUNT(*) AS lake_count
FROM lakes
GROUP BY depth;
```

Количество запросов: 1
Количество задействованных коллекций: 1

Статистика по рейтингу озёр

```
SELECT rating, COUNT(*) AS lake_count
FROM lakes
GROUP BY rating;
```

Количество запросов: 1
Количество задействованных коллекций: 1

Статистика по количеству отзывов

```
SELECT lakes.name, COUNT(reviews.id) AS review_count
FROM lakes
JOIN reviews ON lakes.id = reviews.lake_id
GROUP BY lakes.name;
```

Количество запросов: 1
Количество задействованных коллекций: 1

Поиск всех озер без посещений и без отзывов

```
SELECT *
FROM Lakes
WHERE lake_id NOT IN (SELECT lake_id FROM Visits)
AND lake_id NOT IN (SELECT lake_id FROM Reviews);
```

Количество запросов: 3
Количество задействованных коллекций: 3

Поиск пользователей, которые хотя бы одно озеро посетили, но отзыв не оставили

```
SELECT *
FROM Users
WHERE user_id IN (SELECT user_id FROM Visits)
  AND user_id NOT IN (
      SELECT user_id
      FROM Reviews
      WHERE lake_id IN (SELECT lake_id FROM Visits WHERE user_id = Review
s.user_id)
  );
```

Количество запросов: 4
Количество задействованных коллекций: 3

3.3. Сравнение моделей

Удельный объем данных

В системе нереляционная модель демонстрирует более высокий удельный объем информации по сравнению с реляционной. Так как реляционная модель избегает дублирования, что увеличивает время обработки запросов. Однако нереляционная модель несколько эффективнее по использованию памяти за счет меньшей избыточности данных.

Сравнение по моделям:

- **Пользователь**: 2520 байт - Neo4j; 2040 байт - SQL
- **Отзыв**: 1012 байт - Neo4j; 1046 байт - SQL
- **Озеро**: 3516 байт - Neo4j; 2545 байт - SQL
- **Избыточность**: 1.0067 - Neo4j; 1.0074 - SQL

Запросы для различных сценариев

Сравнение количества запросов для различных сценариев:

- **Регистрация**: 2 запроса - Neo4j; 2 запроса - SQL
- **Авторизация**: 1 запрос - Neo4j; 1 запрос - SQL
- **Добавление в список “Хочу посетить”**: 2 запроса - Neo4j; 2 запроса - SQL

Сравнение по количеству использованных коллекций:

- **Регистрация**: 1 коллекция - Neo4j; 1 коллекция - SQL
- **Авторизация**: 1 коллекция - Neo4j; 1 коллекция - SQL
- **Добавление в список “Хочу посетить”**: 2 коллекции - Neo4j; 1 коллекция - SQL

Обе модели имеют схожую структуру запросов по количеству. В нереляционной базе данных для операций, связанных с взаимодействием между сущностями, задействуется большее количество коллекций. Например, для добавления озера в список “Уже посетил”/“Хочу посетить” задействуются как минимум 2 коллекции (узлы User и Lake).

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1 Краткое описание

Наше приложение представляет собой каталог и агрегатор отзывов на озера РФ, разработанное на языке Java с использованием базы данных Neo4j.

Back-end: Разработан на Spring Boot и содержит следующие модули:

Контроллеры: обрабатывают HTTP-запросы от пользователей и передают их к сервисам.

Сервисы: логический слой, где выполняется бизнес-логика.
Репозитории: обеспечивают доступ к данным, хранящимся в Neo4j.
Front-end: написан с использованием HTML, CSS, JavaScript. Веб-слой использует Thymeleaf для рендеринга HTML-шаблонов, поддерживая пользовательский интерфейс.

Приложение развертывается с использованием Docker Compose.

Контейнер приложения:

Содержит сборку Spring Boot-приложения.
Включает механизм логирования и подключение к базе данных Neo4j.
Собирается из Dockerfile с использованием Maven для сборки JAR-файла.

Контейнер базы данных:

Работает на основе официального образа neo4j.

Контейнеры соединены через общую Docker-сеть (app-network), обеспечивая безопасность и изоляцию. База данных доступна только приложению через внутренний адрес db.

4.2 Использованные технологии

БД: Neo4j.

Back-end: Spring, JavaScript.

Front-end: HTML, CSS, JavaScript.

Контейнеризация: Docker, Docker Compose.

4.3 Схема экранов приложения

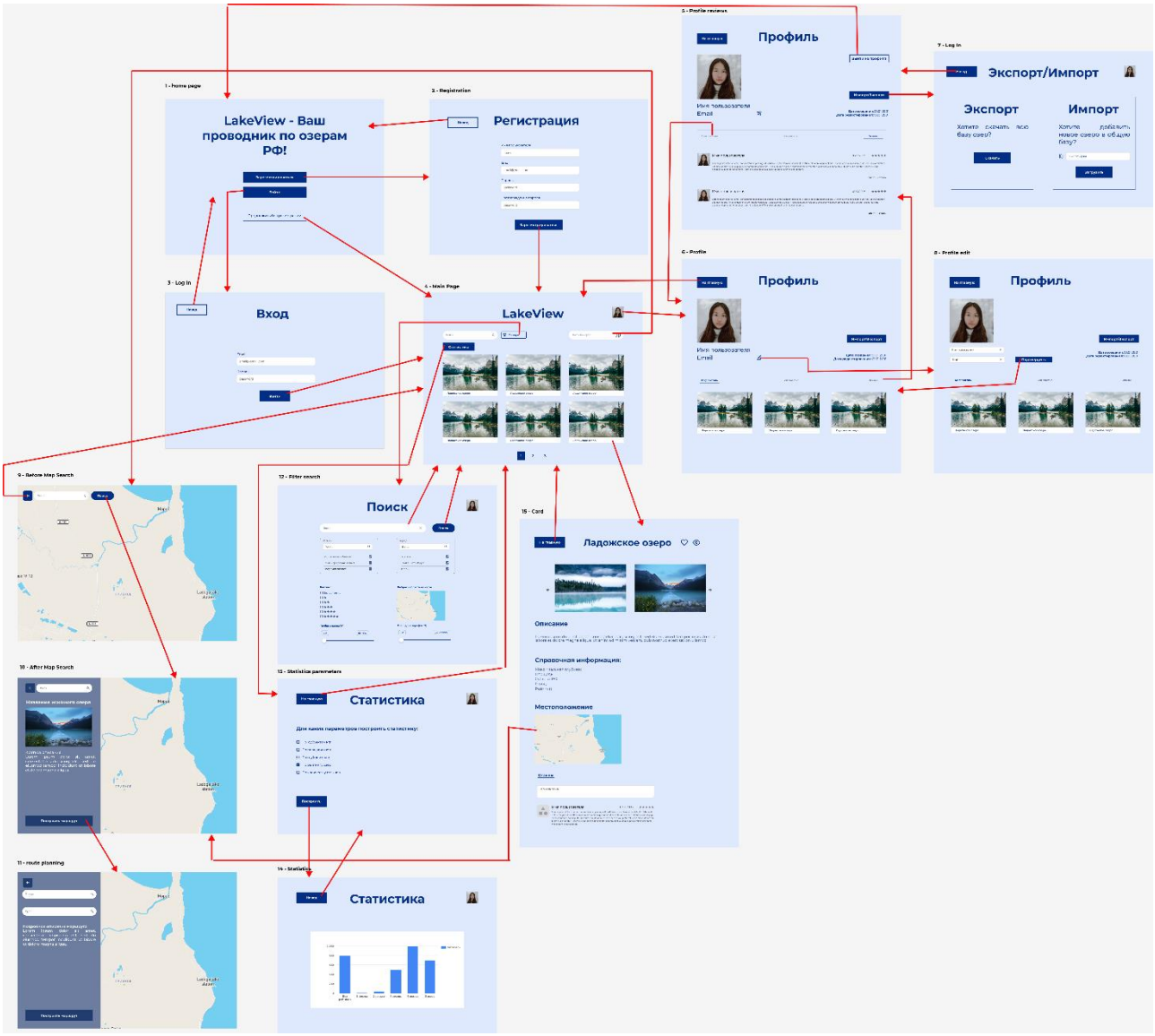


Рисунок 7. Схема экранов приложения

5. ВЫВОДЫ

5.1 Достигнутые результаты

В ходе работы было разработано приложение "LakeView", представляющее собой каталог озер, которое позволяет пользователям добавлять и фильтровать озера, добавлять отзывы к ним. Приложение ориентировано на предоставление пользователям эффективного инструмента для управления и изучения озер.

Также реализована функция просмотра кастомизированной статистики. Для удобства работы с данными доступны функции импорта и экспорта, что позволяет переносить информацию, а также создавать бэкапы для безопасного хранения и восстановления данных.

5.2 Недостатки и пути для улучшения

В текущей версии приложения отсутствует механизм разграничения прав доступа, что создает уязвимость для несанкционированных изменений данных и затрудняет контроль действий пользователей. Для решения этой проблемы можно внедрить систему авторизации с поддержкой ролей, которая позволит ограничивать доступ к функциям приложения в соответствии с уровнями прав пользователей (например, администратор и обычный пользователь). Это повысит уровень контроля и обеспечит защиту данных.

В настоящее время приложение доступно исключительно в веб-формате, что затрудняет его использование на мобильных устройствах. Для расширения возможностей пользователей необходимо разработать мобильную версию приложения, совместимую с платформами iOS и Android. Это обеспечит доступ к функционалу приложения в любое время и в любом месте.

Кроме того, важно улучшить адаптивность существующего веб-приложения, чтобы оно корректно отображалось на устройствах с различными размерами экранов, включая смартфоны, планшеты и компьютеры. Это сделает использование приложения более комфортным.

5.3 Будущее развитие решения

В будущем планируется внедрение разграничением прав доступа для повышения безопасности данных. Также будет создана мобильная версия для iOS и Android, и будет улучшена адаптивность веб-приложения, что сделает его доступным и удобным для пользователей на разных устройствах.

6. ПРИЛОЖЕНИЯ

6.1 Документация по сборке и развертыванию приложения.

1. Склонировать репозиторий с проектом (ссылка указана в списке литературы) и перейти в директорию проекта.
2. Собрать контейнеры приложения командой: *docker-compose build --no-cache*.
3. Запустить контейнеры командой: *docker-compose up*.
4. Открыть приложение в браузере по адресу 127.0.0.1:8080.

6.2 Инструкция для пользователя.

1. Массовый импорт-экспорт данных.

При массовом импорте-экспорте используются файлы формата .json. При массовом экспорте файл lakes.json будет скачан на компьютер пользователя. При массовом импорте пользователь должен выбрать файл формата .json для импорта, озера из которого будут загружены в базу данных.

2. Оставление отзыва на озеро.

Для того, чтобы оставить отзыв на озеро, пользователь должен авторизоваться, после чего ему необходимо перейти на страницу с озером и оставить отзыв в поле для ввода, выбрав нужное количество звезд.

Если поле ввода текста или количество звезд не будет заполнено, приложение сообщит об этом пользователю и отзыв опубликовать не удастся.

3. Кастомизированная статистика.

Для отображения статистики по приложению необходимо на главной странице нажать кнопку “Статистика”, после чего осуществится переход на соответствующую страницу. На этой странице пользователю предоставится возможность выбрать тип статистики. При нажатии на кнопку построения статистики на странице отобразится соответствующая диаграмма.

Система проверит, какой тип статистики выбран, и, если не заполнено одно из необходимых полей (не выбран промежуток дат или асана для статистики), графический интерфейс выведет соответствующие сообщения с предупреждением.

7. ЛИТЕРАТУРА

1. Ссылка на GitHub. - [Электронный ресурс]. - URL: https://github.com/moevm/nosql2h24-lakes.git.
2. Spring Boot. – [Электронный ресурс]. – URL: https://spring.io/projects/spring-boot (дата обращения: 18.12.2024).
3. Neo4j. – [Электронный ресурс]. – URL: https://neo4j.com/ (дата обращения: 18.12.2024).
4. Thymeleaf. – [Электронный ресурс]. – URL: https://www.thymeleaf.org/ (дата обращения: 18.12.2024).
5. Neo4jRepository. – [Электронный ресурс]. – URL: https://docs.spring.io/spring-data/neo4j/docs/current/api/org/springframework/data/neo4j/repository/Neo4jRepository.html (дата обращения: 18.12.2024).
6. Apache Maven. – [Электронный ресурс]. – URL: https://maven.apache.org/ (дата обращения: 18.12.2024).
7. Docker. – [Электронный ресурс]. – URL: https://www.docker.com/ (дата обращения: 18.12.2024).
8. OpenStreetMap. – [Электронный ресурс]. – URL: https://www.openstreetmap.org/#map=3/69.62/-74.90 (дата обращения: 18.12.2024).