

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: Карта транспортной доступности водоемов севера Ленинградской
области

Студент гр. 1303	_____	Токун Г.С.
Студент гр. 1381	_____	Сапожников А. Э.
Студент гр. 1381	_____	Ковалёв П. А.
Преподаватель	_____	Заславский М.М.

Санкт-Петербург,
2024

ЗАДАНИЕ

Студент Токун Г.С.

Студент Сапожников А. Э.

Студент Ковалёв П. А.

Группа 1303

Тема: Карта транспортной доступности водоемов севера Ленинградской области

Исходные данные:

Необходимо реализовать веб-приложение для цифровизации работы пользователей с картой транспортной доступности водоемов севера Ленинградской области.

Содержание пояснительной записки:

«Содержание»

«Введение»

«Сценарии использования»

«Модель данных»

«Разработанное приложение»

«Выводы»

«Приложения»

«Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 05.09.2024

Дата сдачи реферата: 24.12.2024

Дата защиты реферата: 24.12.2024

Студент	_____	Токун Г.С.
Студент	_____	Сапожников А. Э.
Студент	_____	Ковалёв П. А.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках проекта разработана платформа для управления и отображения карты доступности, предоставляющая пользователям интерактивный доступ к данным о доступности различных мест и маршрутов. Система позволяет пользователям исследовать, добавлять и редактировать точки на карте, создавать маршруты и получать статистику об использовании данных маршрутов. Также реализован функционал обращения в поддержку для связи с администрацией. Для организации данных использована нереляционная база данных MongoDB, что обеспечивает гибкость и масштабируемость системы. Платформа использует Neo4j для графового представления связей и маршрутов, что позволяет эффективно управлять сложными запросами и взаимодействиями между объектами.

SUMMARY

The project developed a platform for managing and displaying an accessibility map, providing users with interactive access to information about the accessibility of various locations and routes. The system allows users to explore, add, and edit points on the map, create routes, and receive statistics on the usage of these routes. Additionally, a support request feature is implemented for communication with the administration. The non-relational MongoDB database is used to organize data, ensuring flexibility and scalability of the system. The platform utilizes Neo4j for graph representation of relationships and routes, facilitating efficient management of complex queries and interactions between objects.

СОДЕРЖАНИЕ

1.	Введение	6
1.1.	Актуальность 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	10
2.3.	Вывод	15
3.	Модель данных	16
3.1.	Нереляционная модель данных	16
3.2.	Реляционная модель данных	21
3.3.	Сравнение моделей	25
4.	Разработанное приложение	26
5.	Выводы	29
6.	Приложения	30
7.	Литература	31

1. ВВЕДЕНИЕ

1.1. Актуальность проблемы

Актуальность проблемы В современном мире, где внимание к доступности и инклюзивности окружающего пространства растет, возникает необходимость в создании инструментов для удобного взаимодействия с городскими объектами. Платформа для управления картой доступности помогает пользователям оценивать и планировать маршруты с учетом особенностей инфраструктуры, что особенно важно для людей с ограниченными возможностями здоровья. Такое решение способствует повышению качества жизни и независимости этой категории граждан.

1.2. Постановка задачи

1. Интерактивное отображение доступности объектов и маршрутов на карте;
2. Возможность пользователей добавлять и редактировать точки доступности;
3. Создание и управление маршрутами с учетом индивидуальных потребностей пользователей;
4. Функционал для обратной связи и поддержки пользователей для улучшения и актуализации данных.

1.3. Предлагаемое решение

Предлагаемое решение Разработка веб-платформы, данные о точках доступности, пользовательские маршруты и обратная связь будут храниться в нереляционной базе данных MongoDB, а взаимосвязи между объектами будут управляться с помощью графовой базы данных Neo4j для эффективной обработки сложных запросов и аналитики.

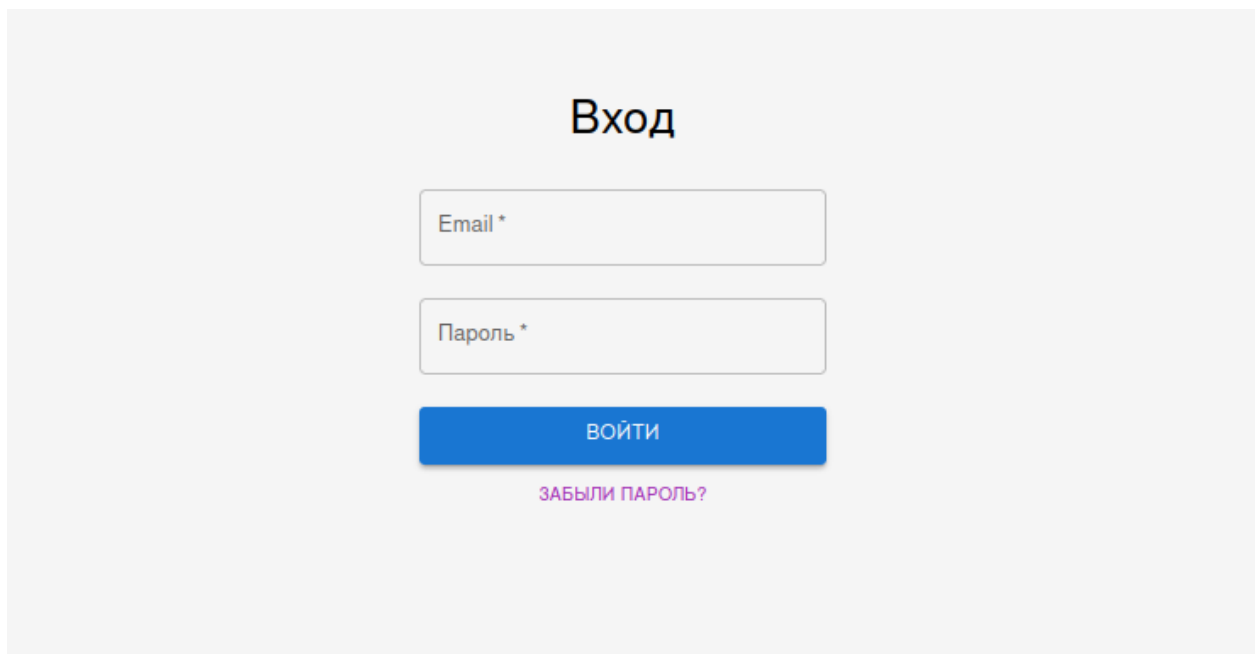
1.4. Качественные требования к решению

Качественные требования к решению Платформа должна быть интуитивно понятной и доступной для всех категорий пользователей, обладать высокой производительностью и безопасностью.

2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

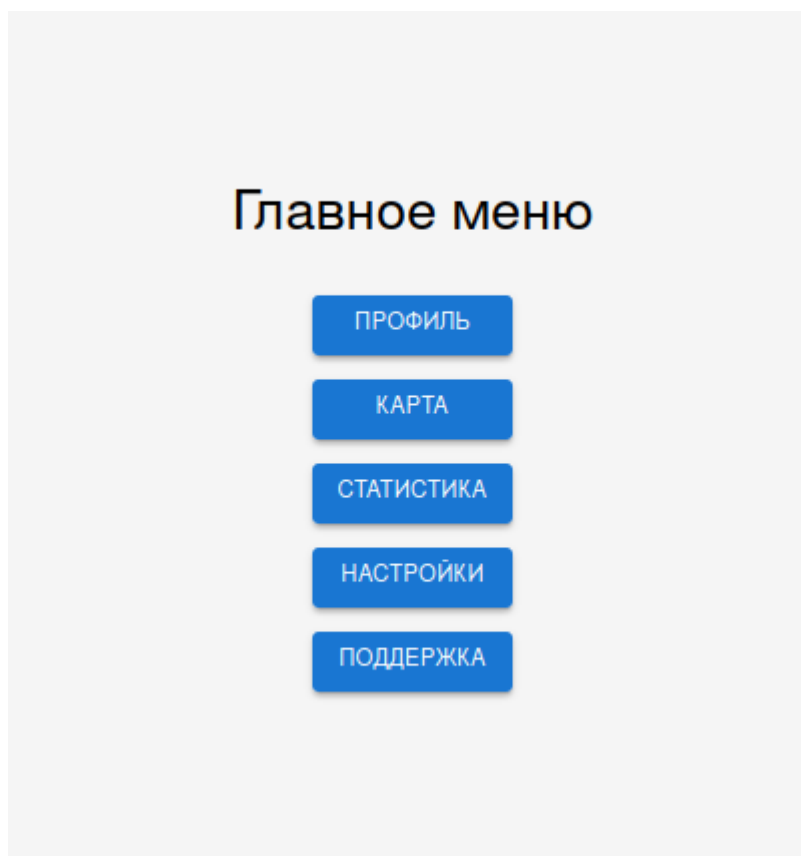
2.1. Макет UI

Ниже представлены макеты страниц приложения.



The image shows a login form titled "Вход" (Login) centered on a light gray background. Below the title are two input fields: "Email *" and "Пароль *" (Password *). Below these fields is a blue button labeled "ВОЙТИ" (Login). Underneath the button is a link labeled "ЗАБЫЛИ ПАРОЛЬ?" (Forgot password?) in purple text.

Рисунок 1. Форма входа



The image shows a main menu titled "Главное меню" (Main menu) centered on a light gray background. Below the title are five blue buttons stacked vertically, labeled "ПРОФИЛЬ" (Profile), "КАРТА" (Map), "СТАТИСТИКА" (Statistics), "НАСТРОЙКИ" (Settings), and "ПОДДЕРЖКА" (Support).

Рисунок 2. Главное меню

[НАЗАД](#)

Профиль пользователя

Имя

Иван Иванов

Email

ivan@example.com

Телефон

+7 (900) 123-45-67

РЕДАКТИРОВАТЬ

Мои маршруты:

Маршрут 1 - 15 декабря 2024

Маршрут 2 - 20 декабря 2024

Маршрут 3 - 25 декабря 2024

Рисунок 3. Просмотр профиля и его завершенных маршрутов

Поиск

Введите название

Рейтинг

ПРИМЕНИТЬ ФИЛЬТРЫ

Результаты

Центральный маркер - Центр Санкт-Петербурга (Рейтинг: 5)

Маркер на юге - Южный район (Рейтинг: 4)

Маркер на севере - Северный район (Рейтинг: 3)

Рисунок 4. Просмотр маршрутов на карте

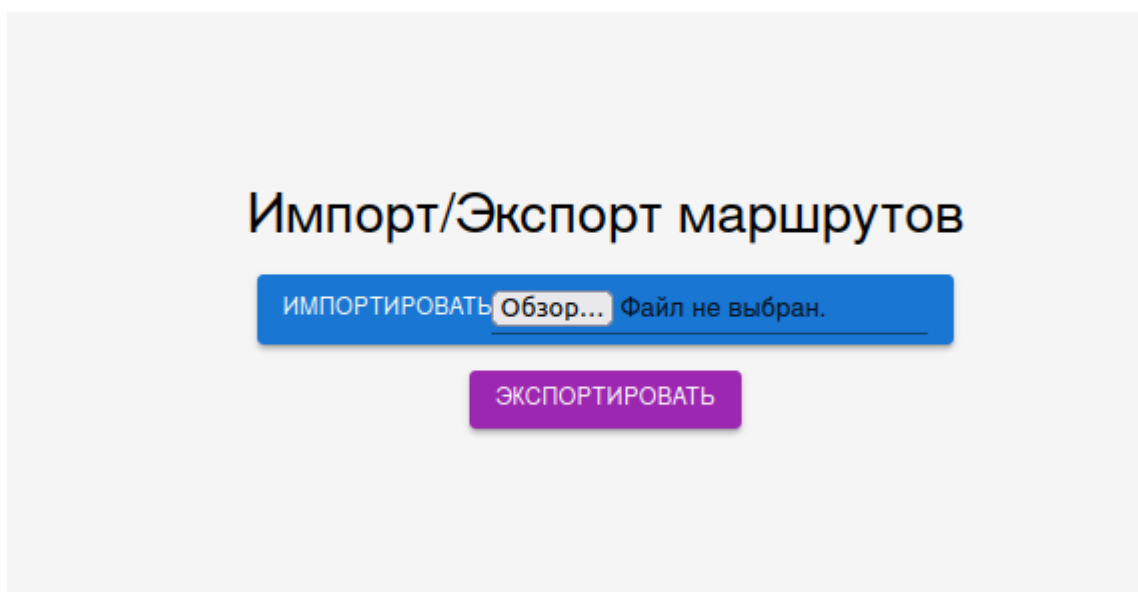


Рисунок 5. Возможность импорта и экспорта маршрутов

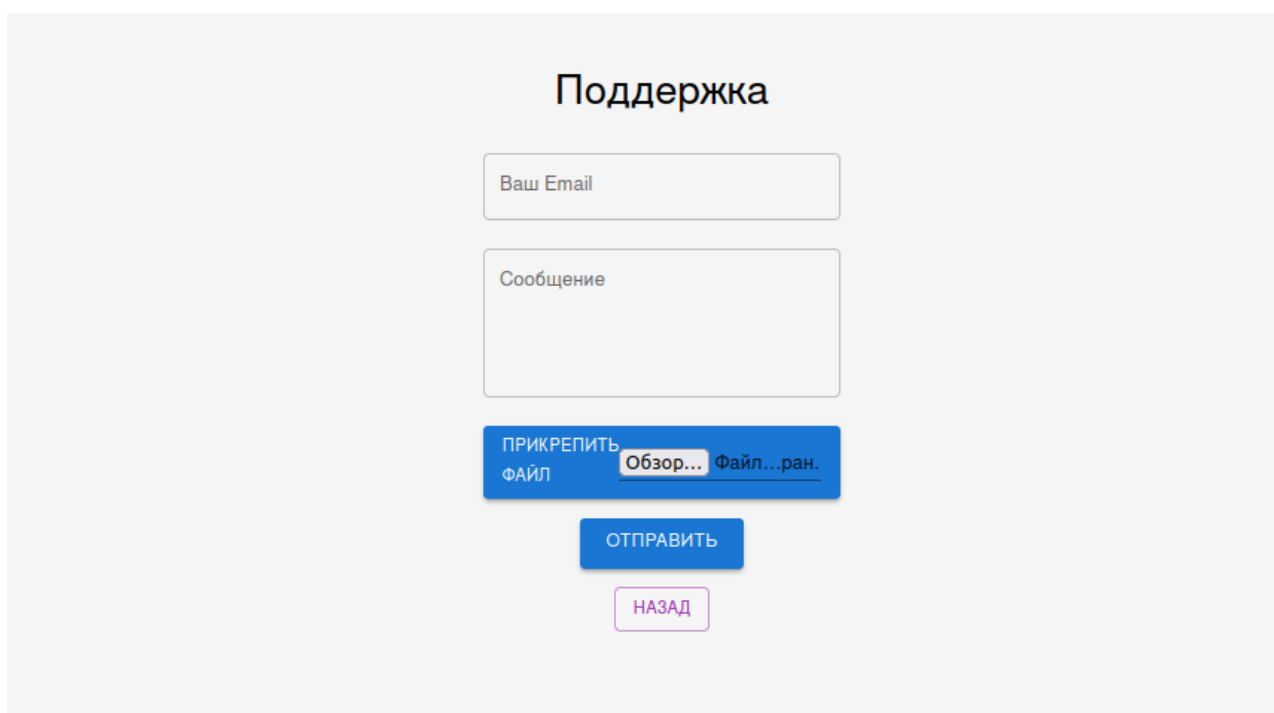


Рисунок 6. Возможность обращения в поддержку с прикреплением файлов

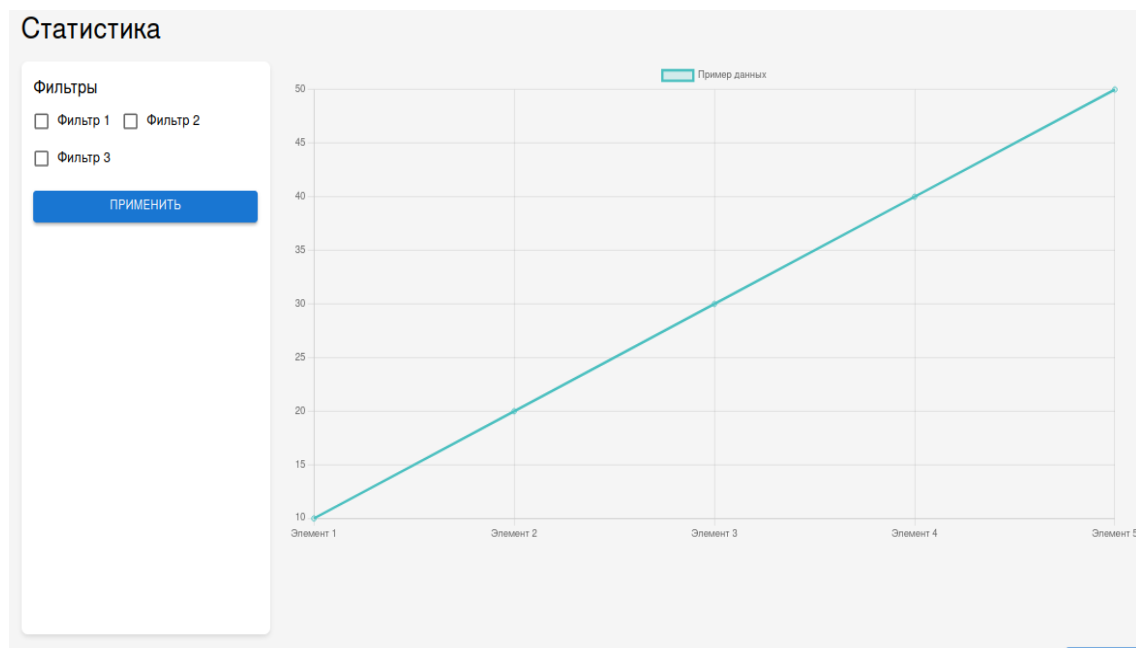


Рисунок 7. Просмотр статистики администратором

2.2. Сценарии использования

1. Сценарий использования: Импорт данных

- **Действующее лицо:** Администратор
- **Предусловие:** Администратор авторизован в системе.

Основной сценарий:

1. Администратор открывает веб-приложение и авторизуется.
2. В разделе администрирования выбирает опцию "Импорт данных".
3. Администратор выбирает файл с данными для загрузки.
4. Приложение проверяет структуру и формат загружаемого файла.
5. Приложение импортирует данные из файла в базу данных.
6. Приложение уведомляет администратора об успешном завершении процесса импорта.

Альтернативный сценарий:

- Если файл имеет неправильный формат или содержит ошибки:
 1. Приложение выводит сообщение об ошибке, указывая на неверный формат файла или ошибки в данных.
 2. Администратор получает возможность выбрать и загрузить другой файл.

2. Сценарий использования: Построение маршрута

- Действующее лицо: Пользователь
- Предусловие: Пользователь авторизован в системе.

Основной сценарий:

1. Пользователь заходит в раздел "Построение маршрута".
2. Приложение отображает карту с возможными точками для построения маршрута.
3. Пользователь выбирает начальную и конечную точки маршрута.
4. Приложение предлагает указать параметры маршрута, такие как предпочтительный тип пути (например, доступный для колясок или пешеходный).
5. Пользователь вводит все необходимые данные и нажимает "Построить маршрут".
6. Приложение обрабатывает запрос, строит маршрут с учетом указанных параметров и отображает его на карте.
7. Пользователь может сохранить маршрут в своем профиле или начать навигацию.

Альтернативный сценарий:

- Если маршрут не может быть построен из-за отсутствия доступных путей между выбранными точками:
 1. Приложение уведомляет пользователя о невозможности построения маршрута и предлагает изменить точки или параметры маршрута.

Результат: Маршрут построен согласно указанным параметрам пользователя и доступен для просмотра и использования в системе.

3. Сценарий использования: Просмотр сохраненных маршрутов

- Действующее лицо: Пользователь
- Предусловие: Пользователь авторизован в системе и имеет сохраненные маршруты.

Основной сценарий:

1. Пользователь заходит в раздел "Мои маршруты".
2. Приложение отображает список всех сохраненных пользователем маршрутов.
3. Пользователь выбирает интересующий маршрут для просмотра деталей.
4. Приложение показывает детальную карту маршрута с указанием всех точек, а также информацию о длине маршрута, предполагаемом времени прохождения и доступности для разных категорий пользователей.

5. Пользователь может выбрать опцию "Начать навигацию" для использования маршрута в реальном времени или "Редактировать маршрут", если необходимо внести изменения.
6. Приложение обновляет информацию в зависимости от выбора пользователя и предоставляет соответствующие инструменты управления.

Альтернативный сценарий:

- Если у пользователя не сохранено ни одного маршрута:
 1. Приложение уведомляет пользователя о том, что сохраненных маршрутов нет.
 2. Предлагает перейти в раздел "Построение маршрута" для создания нового.

Результат: Пользователь успешно просматривает или управляет сохраненными маршрутами в соответствии с его потребностями.

4. Сценарий использования: Построение нескольких маршрутов

- Действующее лицо: Пользователь
- Предусловие: Пользователь авторизован в системе.

Основной сценарий:

1. Пользователь заходит в раздел "Планировщик маршрутов" на странице приложения.
2. Приложение предлагает опции для построения маршрутов, включая выбор начальной и конечной точек, а также возможность добавления промежуточных точек.
3. Пользователь вводит необходимые данные для каждой точки маршрута и выбирает предпочтительные параметры доступности.
4. Приложение обрабатывает запрос и строит несколько вариантов маршрутов, учитывая заданные критерии.
5. Приложение отображает возможные маршруты на карте с детальной информацией о каждом маршруте, включая расстояние, предполагаемое время в пути, и уровень доступности.
6. Пользователь может выбрать наиболее подходящий маршрут или изменить параметры поиска для нового построения.

Альтернативный сценарий:

- Если система не может найти маршруты, соответствующие всем указанным критериям:
 1. Приложение информирует пользователя о невозможности построить маршрут с текущими параметрами.

2. Предлагает пользователю изменить или упростить некоторые критерии для успешного построения маршрута.

- **Результат:** Пользователь получает информацию о различных возможных маршрутах, что позволяет ему эффективно планировать перемещения с учетом своих нужд и предпочтений.

5. Сценарий использования: Просмотр статистики по пройденным маршрутам

- Действующее лицо: Пользователь
- Предусловие: Пользователь авторизован в системе.

Основной сценарий:

1. Пользователь заходит в раздел "Статистика маршрутов".
2. Приложение отображает список всех маршрутов, пройденных пользователем за определённый период.
3. Пользователь может выбрать период для фильтрации, например, за последнюю неделю или месяц.
4. Приложение отображает статистику по каждому маршруту, включая дату прохождения, длину, продолжительность и количество шагов.
5. Пользователь просматривает детализированную информацию о своих активностях и достижениях.

Результат: Пользователь получает полную статистику о своих пройденных маршрутах за выбранный период, что помогает ему отслеживать свои успехи и планировать тренировки.

6. Сценарий использования: Полный доступ к информации о пользователях и маршрутах

- Действующее лицо: Администратор
- Предусловие: Администратор авторизован в системе.

Основной сценарий:

1. Администратор заходит в раздел "Пользователи и маршруты".
2. Приложение отображает список всех пользователей и их маршруты.
3. Администратор выбирает конкретного пользователя для просмотра детальной информации.
4. Приложение отображает полную информацию о выбранном пользователе, включая личные данные, список созданных и пройденных маршрутов, а также их детали и статистику.
5. Администратор просматривает данные.

Результат: Администратор получает доступ к полной информации о пользователях и их активностях в системе.

7. **Сценарий использования:** Просмотр информации о точках доступности
- Действующее лицо: Администратор
 - Предусловие: Администратор авторизован в системе.

Основной сценарий:

1. Администратор заходит в раздел "Информация о доступности".
2. Приложение отображает данные о всех точках доступности, включая их расположение, тип доступности (например, доступно для инвалидов колясок, наличие лифтов и т.д.), и время работы.
3. Администратор может фильтровать точки по различным критериям, таким как район, тип доступности и уровень обслуживания.
4. Администратор просматривает и анализирует данные.

Результат: Администратор получает подробную информацию о точках доступности, что помогает в планировании улучшений инфраструктуры и обслуживания пользователей.

8. **Сценарий использования:** Экспорт информации о маршрутах
- Действующее лицо: Администратор
 - Предусловие: Администратор авторизован в системе.

Основной сценарий:

1. Администратор открывает веб-приложение и авторизуется.
2. В разделе администрирования выбирает опцию "Экспорт маршрутов".
3. Приложение предлагает выбрать формат данных для экспорта (например, JSON, CSV) и определить, какие данные о маршрутах будут включены (например, координаты, описания, уровень доступности).
4. Администратор настраивает параметры экспорта и подтверждает выбор.
5. Приложение генерирует файл с данными в выбранном формате.
6. Администратор загружает файл на своё устройство.

Альтернативный сценарий:

- Если возникают проблемы с доступом к данным или с генерацией файла:
 1. Приложение выводит сообщение об ошибке, указывая на проблему доступа к базе данных или ошибку при формировании файла.
 2. Администратор может попробовать изменить параметры экспорта или повторить попытку позже.

Результат: Администратор успешно экспортирует данные о маршрутах из базы данных, что позволяет использовать их для анализа, отчетности или интеграции с другими системами.

2.3 Вывод:

Для разработанной системы управления картой доступности также характерно преобладание операций чтения над записью данных. Хотя пользователи и администраторы могут вносить изменения в информацию о маршрутах и точках доступности, основная деятельность связана с просмотром и анализом существующих данных.

3. МОДЕЛЬ ДАННЫХ

3.1. Нереляционная модель данных

Графическое представление модели

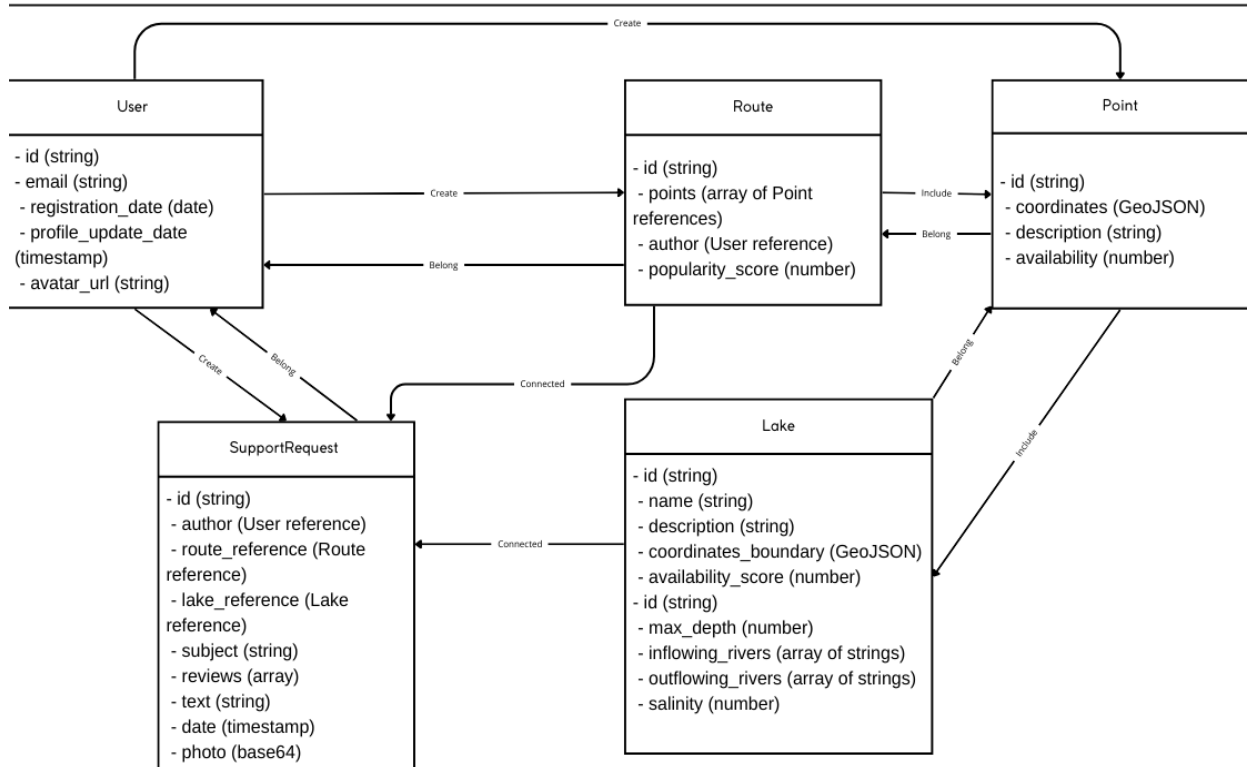


Рисунок 7. Нереляционная модель данных

Описание назначений коллекций, типов данных и сущностей:

User - Хранение информации о пользователях.

- id (string): Уникальный идентификатор пользователя.
- email (string): Электронная почта пользователя.
- created_at (timestamp): Дата регистрации пользователя.
- updated_at (timestamp): Дата и время последнего обновления профиля.
- avatar_url (string): Ссылка на аватар пользователя.

Point - Хранение данных о точках маршрутов.

- id (string): Уникальный идентификатор точки.
- coordinates (GeoJSON): Координаты точки на карте.
- description (string): Описание точки.
- availability (number): Уровень доступности.

Route - Маршруты, составленные пользователями.

- `id` (string): Уникальный идентификатор маршрута.
- `points` (array of references): Массив ссылок на точки маршрута (Point).
- `author` (reference to User): Идентификатор автора маршрута.
- `popularity_score` (number): Рейтинг маршрута на основе обращений в поддержку и популярности.

Lake - Озера, которые являются точками интереса для маршрутов.

- `id` (string): Уникальный идентификатор озера.
- `name` (string): Название озера.
- `description` (string): Описание озера.
- `coordinates_boundary` (GeoJSON): Координаты границ озера.
- `availability_score` (number): Оценка доступности озера (количество доступных к нему маршрутов).
- `max_depth` (number): Максимальная глубина озера.
- `inflowing_rivers` (array of strings): Список названий рек, впадающих в озеро.
- `outflowing_rivers` (array of strings): Список названий рек, вытекающих из озера.
- `salinity` (number): Уровень солености озера.

SupportTicket - Запросы в поддержку от пользователей.

- `id` (string): Уникальный идентификатор обращения.
- `author` (reference to User): Автор обращения.
- `route_reference` (reference to Route): Связь с маршрутом, если обращение касается маршрута.
- `lake_reference` (reference to Lake): Связь с озером, если обращение касается озера.
- `subject` (string): Тема обращения.
- `text` (string): Текст отзыва.
- `created_at` (timestamp): Дата обращения.
- `photo` (base64): Фото, закодированное в base64.

Оценка объема информации, хранимой в модели

- Пусть:
 - (N) — количество пользователей.
 - В среднем на один маршрут приходится 10 точек.
 - Сообщения в поддержку пишет каждый 10-й пользователь.

- Каждый пользователь совершает около 10 маршрутов.
- Озер всего 100.

Пользователь (User)

- id (строка, 8 байт)
- email (строка, 50 байт)
- дата регистрации (дата, 8 байт)
- дата обновления профиля (timestamp, 8 байт)
- аватар (строка, 100 байт)
- Итого:
- $(8 + 50 + 8 + 8 + 100 = 174)$ байта на пользователя

Точка (Point)

- id (строка, 8 байт)
- координаты (GeoJSON, 16 байт)
- описание (строка, 100 байт)
- доступность (целое число, 4 байта)
- Итого:
- $(8 + 16 + 100 + 4 = 128)$ байт на точку

Маршрут (Route)

- id (строка, 8 байт)
- список точек (массив ссылок на узлы точек, в среднем 10 ссылок, по 8 байт каждая): $(10 \times 8 = 80)$ байт
- автор (ссылка на User, 8 байт)
- рейтинг (число, 4 байта)
- Итого:
- $(8 + 80 + 8 + 4 = 100)$ байт на маршрут

Озеро (Lake)

- id (строка, 8 байт): Уникальный идентификатор озера.
- name (строка, 50 байт): Название озера.
- description (строка, 200 байт): Описание озера.

- `coordinates_boundary` (GeoJSON, 50 байт): Координаты границ озера.
- `availability_score` (число, 4 байта): Оценка доступности озера.
- `max_depth` (число, 4 байта): Максимальная глубина озера.
- `inflow_rivers` (строка[], 100 байт): Список впадающих рек (примерное значение для массива строк).
- `outflow_rivers` (строка[], 100 байт): Список вытекающих рек.
- `salinity` (число, 4 байта): Соленость озера.
- **Итого для Lake:** $8 + 50 + 200 + 50 + 4 + 4 + 100 + 100 + 4 = 520$ { байт}

Обращение в поддержку (SupportRequest)

- `id` (строка, 8 байт)
- `автор` (ссылка на User, 8 байт)
- `связь с маршрутом` (ссылка на Route, 8 байт)
- `связь с озером` (ссылка на Lake, 8 байт)
- `тема` (строка, 50 байт)
- `отзывы` (массив объектов):
 - `текст` (строка, 200 байт)
 - `дата` (timestamp, 8 байт)
 - `фото` (base64, 2 796 202 байта)

Общий объем памяти для связей

- Теперь давайте подытожим объем памяти для всех указанных связей:
 $37+37+37+37+37+37+39+39+37+37+37 = 398$ байт

Итого для расчета на пользователя

1. **Пользователь (User):** 174 байта
 2. **Точки (Point):** 12800 байт (10 точек по 128 байт в 10 разных маршрутах)
 3. **Маршруты (Route):** 1000 байт (10 маршрутов по 100 байт)
 4. **Обращения в поддержку (SupportRequest):** 2,796,492 байт делим на 10
 5. ****Связи:** 398 байт
 6. **Озера:** 520 байт
- **Итого:**
 - **Общий объем данных на одного пользователя** = $174 + 12800 + 1000 + 279,649 + 398 + 520 \cdot 100 = 346,047$ байта = 0,346 мбайт

Примеры запросов

1. Получение всех пользователей

```
MATCH (u:User)
RETURN u.id AS userId, u.email AS email, u.registration_date AS
registrationDate
```

2. Поиск точки по идентификатору

```
MATCH (p:Point {id: 'point001'})
RETURN p.id AS pointId, p.coordinates AS coordinates, p.description AS
description, p.availability AS availability
```

3. Получение всех маршрутов с их точками

```
MATCH (r:Route)-[:CONTAINS]->(p:Point)
RETURN r.id AS routeId, collect(p.id) AS points
```

4. Получение всех точек, доступных для маршрута

```
MATCH (r:Route {id: 'route001'})-[:CONTAINS]->(p:Point)
RETURN p.id AS pointId, p.description AS description
```

5. Добавление нового пользователя

```
CREATE (u:User {id: 'user003', email: 'user3@example.com',
registration_date: '2024-02-03'})
RETURN u
```

6. Добавление новой точки

```
CREATE (p:Point {id: 'point003', coordinates: '59.9342, 30.3351',
description: 'Невский проспект', availability: ['Доступно']})
RETURN p
```

7. Создание маршрута с точками

```
CREATE (r:Route {id: 'route003'})
WITH r
CREATE (p1:Point {id: 'point001'})-[:CONTAINS]->(r)
CREATE (p2:Point {id: 'point002'})-[:CONTAINS]->(r)
RETURN r
```

8. Получение обращений в поддержку по пользователю

```
MATCH (u:User)-[:MADE]->(sr:SupportTicket)
WHERE u.id = 'user001'
RETURN sr.id AS supportRequestId, sr.subject AS subject, sr.reviews AS
reviews
```

9. Получение статистики обращений в поддержку

```
MATCH (sr:SupportTicket)
RETURN COUNT(sr) AS totalSupportTickets
```

10. Обновление темы обращения в поддержку

```
MATCH (sr:SupportTicket {id: 'support001'})
SET sr.subject = 'Обновленная тема обращения'
RETURN sr
```

11. Поиск самых популярных маршрутов и отзывов (по количеству обращений в поддержку)

```
*MATCH (route:Route)<-[:route_reference]-(request:SupportTicket)
*RETURN route.id AS RouteID, COUNT(request) AS SupportTicketsCount
*ORDER BY SupportTicketsCount DESC
*LIMIT 10;
```

12. Самые длинные маршруты

```
MATCH (route:Route)
WITH route, SIZE(route.points) AS pointCount
RETURN route.id AS RouteID, pointCount
ORDER BY pointCount DESC
LIMIT 10;
```

3.2. Аналог модели данных для SQL СУБД

Графическое представление модели

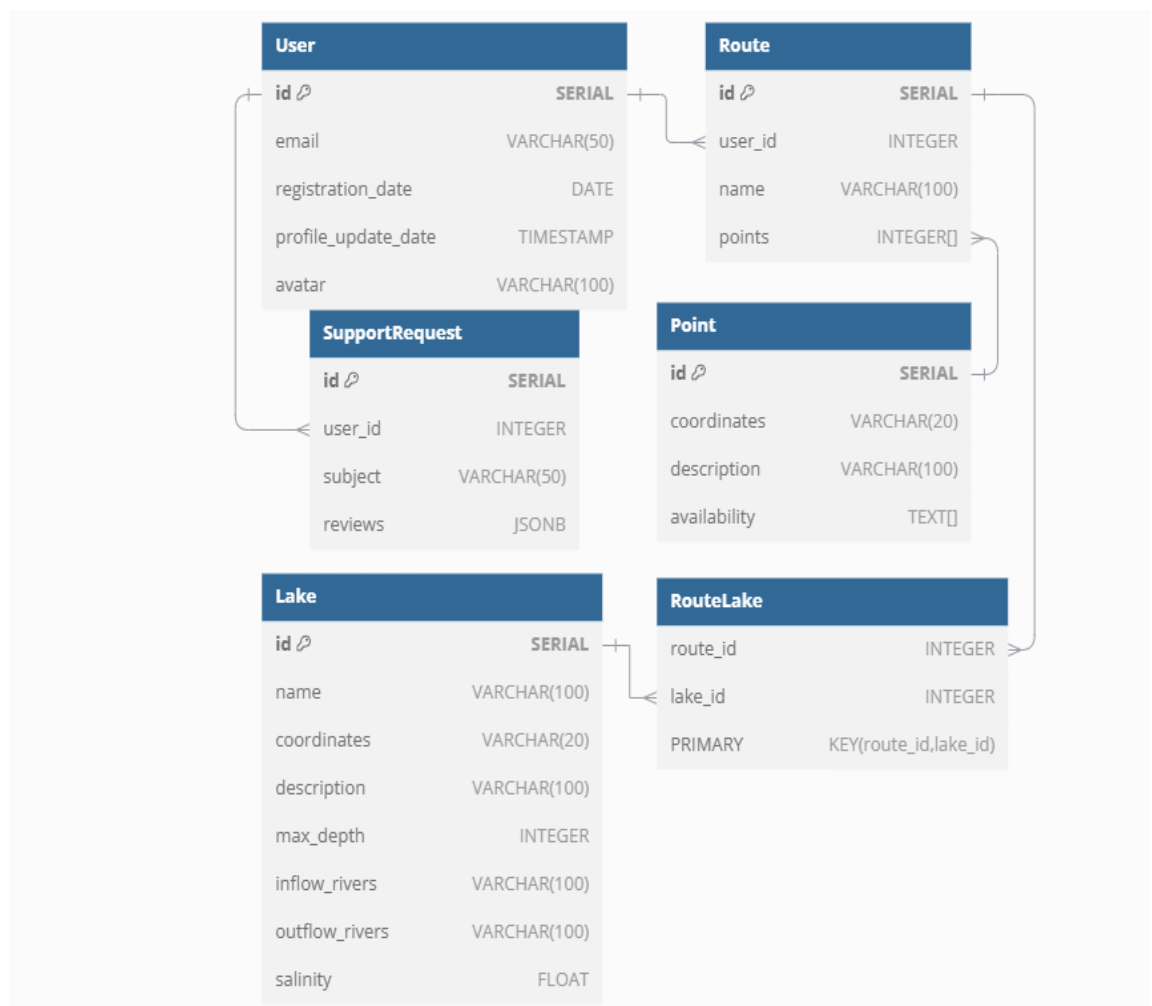


Рисунок 8. Реляционная модель данных

Описание коллекций

1. Пользователь (User)

- **Назначение:** Хранение информации о пользователях, которые могут создавать маршруты и оставлять обращения в поддержку.
- **Поля:**
 - **id:** `SERIAL` — Уникальный идентификатор пользователя, занимает 4 байта.
 - **email:** `VARCHAR(50)` — Электронная почта пользователя, занимает до 50 байт.
 - **created_at:** `TIMESTAMP` — Дата регистрации пользователя, занимает 8 байт.
 - **updated_at:** `TIMESTAMP` — Дата последнего обновления профиля, занимает 8 байт.
 - **avatar:** `VARCHAR(100)` — Ссылка на аватар пользователя, занимает до 100 байт.

2. Точка (Point)

- **Назначение:** Хранение информации о точках, которые могут быть добавлены в маршруты.
- **Поля:**
 - **id:** `SERIAL` — Уникальный идентификатор точки, занимает 4 байта.
 - **coordinates:** `VARCHAR(20)` — Координаты точки на карте, занимает до 20 байт.
 - **description:** `VARCHAR(100)` — Описание точки, занимает до 100 байт.
 - **availability:** `TEXT[]` — Список атрибутов доступности, массив строк (занимает переменное количество байт).

3. Маршрут (Route)

- **Назначение:** Хранение информации о маршрутах, созданных пользователями, включая точки, которые входят в маршрут.
- **Поля:**
 - **id:** `SERIAL` — Уникальный идентификатор маршрута, занимает 4 байта.
 - **user_id:** `INTEGER` — Идентификатор пользователя, создающего маршрут (ссылка на таблицу User), занимает 4 байта.
 - **name:** `VARCHAR(100)` — Название маршрута, занимает до 100 байт.
 - **points:** `INTEGER[]` — Список идентификаторов точек (ссылки на таблицу Point), массив целых чисел (занимает переменное количество байт).

4. Запрос в поддержку (SupportTicket)

- **Назначение:** Хранение информации о запросах пользователей в службу поддержки.
- **Поля:**
 - **id:** SERIAL — Уникальный идентификатор обращения, занимает 4 байта.
 - **user_id:** INTEGER — Идентификатор пользователя, отправляющего обращение (ссылка на таблицу User), занимает 4 байта.
 - **subject:** VARCHAR(50) — Тема обращения, занимает до 50 байт.
 - **reviews:** JSONB — Отзывы по обращению, хранящиеся в формате JSON (занимает переменное количество байт).

5. Озеро (Lake)

- **Назначение:** Хранение информации об озёрах, которые могут быть связаны с маршрутами.
- **Поля:**
 - **id:** SERIAL — Уникальный идентификатор озера, занимает 4 байта.
 - **name:** VARCHAR(100) — Название озера, занимает до 100 байт.
 - **coordinates_boundary:** VARCHAR(50) — Координаты границ озера, занимает до 50 байт.
 - **description:** VARCHAR(200) — Описание озера, занимает до 200 байт.
 - **availability_score:** INTEGER — Оценка доступности озера, занимает 4 байта.
 - **max_depth:** INTEGER — Максимальная глубина озера, занимает 4 байта.
 - **inflow_rivers:** TEXT — Список впадающих рек, текстовый формат (размер зависит от содержимого).
 - **outflow_rivers:** TEXT — Список вытекающих рек, текстовый формат (размер зависит от содержимого).
 - **salinity:** INTEGER — Соленость озера, занимает 4 байта.

6. Связь между маршрутом и озером (RouteLake)

- **Назначение:** Хранение информации о связи между маршрутами и озёрами.
- **Поля:**
 - **route_id:** INTEGER — Идентификатор маршрута (ссылка на таблицу Route).
 - **lake_id:** INTEGER — Идентификатор озера (ссылка на таблицу Lake).
 - **PRIMARY KEY (route_id, lake_id)** — Составной первичный ключ.

Оценка объема информации, хранимой в модели

- Вычислим средний размер объектов каждой сущности:
- **User:** (4 + 50 + 8 + 8 + 100 = 170) байт
- **Point:** (4 + 20 + 100 + 30 = 154) байта

- **Route:** $(4 + 4 + 100 + 96 = 204)$ байта
- **SupportTicket:** $(4 + 4 + 50 + 2796202 = 2796260)$ байт
- **Lake:** $(4 + 100 + 20 + 100 + 4 + 4 + 100 + 100 + 4 = 436)$ байта
- **RouteLake:** $(4 + 4 = 8)$ байт
- Пусть в среднем на один маршрут приходится 10 точек, а также сообщения в поддержку пишет каждый 10 человек, ну и примерно каждый пользователь совершает 10 маршрутов и всего 100 озер.
- При количестве пользователей, равном N, получаем: Итог = $326720 \cdot N$ байт

Примеры запросов

Текст запросов:

Вставка данных

- INSERT INTO Users (email, registration_date) VALUES ('newuser@example.com', NOW());
- INSERT INTO Points (coordinates, description, availability) VALUES ('59.9343,30.3351', 'Площадь Восстания', '["доступен для пешеходов", "доступен для велосипедистов"]');
- INSERT INTO Routes (points) VALUES (ARRAY[1, 2]);
- INSERT INTO SupportRequests (support_request_id, text, date, photo) VALUES (1, 'Маршрут не отображается', NOW(), 'base64image...');

Выборка данных

- SELECT * FROM Users;
- SELECT * FROM Points WHERE availability @> '["доступен для пешеходов"]';
- SELECT r.id AS route_id, p.id AS point_id, p.description FROM Routes r JOIN unnest(r.points) AS point_id ON TRUE JOIN Points p ON p.id = point_id;

Обновление данных

- UPDATE Users SET email = 'updateduser@example.com' WHERE id = 1;
- UPDATE Points SET description = 'Новая площадь Восстания' WHERE id = 1;

Удаление данных

- DELETE FROM Points WHERE id = 1;
- DELETE FROM SupportTickets WHERE id = 1;

3.3. Сравнение моделей

Удельный объем информации

- В реляционной модели данные о пользователях, точках, маршрутах и озерах распределены по различным таблицам, что позволяет избежать дублирования данных и снизить общий объем за счет нормализации. В нереляционной модели MongoDB, используемая денормализация упрощает доступ к данным при чтении, но увеличивает объем хранимой информации из-за дублирования объектов в разных коллекциях.

Примеры запросов по юзкейсам

- **Получение списка маршрутов для пользователя с указанием точек и озер:**
 - **Реляционная модель:** Требуется выполнить несколько JOIN-запросов между таблицами Route, Point, Lake, и User для сбора всех необходимых данных. Это требует сложной обработки, но индексирование и оптимизация SQL-запросов могут ускорить этот процесс.
 - **Нереляционная модель:** Запрос к одной коллекции Routes, где уже содержится вся необходимая информация о точках и озерах, благодаря денормализации. Это упрощает и ускоряет запросы на чтение.
- **Добавление новой точки доступа и связывание ее с маршрутом:**
 - **Реляционная модель:** Создание записи в таблице Point и обновление Route для добавления ссылки на новую точку. Это гарантирует целостность данных и управляемость связей.
 - **Нереляционная модель:** Добавление новой точки непосредственно в документ Route в коллекции Routes, что может потребовать обновления большого количества документов при изменении маршрута.
- **Поиск всех пользователей, выбравших определенный маршрут:**
 - **Реляционная модель:** Необходимость выполнения сложного запроса с JOIN между таблицами User, Route, и вспомогательными таблицами для связей.
 - **Нереляционная модель:** Простой запрос к коллекции Routes, где пользователи, выбравшие маршрут, уже ассоциированы с документом маршрута.

Количество задействованных коллекций/таблиц

- **Список маршрутов для пользователя:** Реляционная модель (4 таблицы: User, Route, Point, Lake) vs. Нереляционная модель (1 коллекция: Routes)

- **Добавление точки к маршруту:** Реляционная модель (2 таблицы: Point, Route) vs. Нереляционная модель (1 коллекция: Routes)
- **Поиск пользователей по маршруту:** Реляционная модель (3 таблицы: User, Route, вспомогательная таблица связей) vs. Нереляционная модель (1 коллекция: Routes)
- Такой подход к анализу вашей модели данных помогает выявить ключевые различия между реляционной и нереляционной моделями и их влияние на архитектуру и производительность вашего приложения.

3) Вывод:

В заключение, выбор между реляционной и нереляционной моделями данных зависит от требований к производительности, гибкости, объему информации. Реляционная модель гарантирует целостность данных, а Нереляционная модель предлагает гибкость и производительность при больших объемах данных, что идеально для современных высоконагруженных приложений. Это облегчит управление данными без необходимости жесткой схемы и снизит сложность поддержки приложения в долгосрочной перспективе.

4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

4.1 Краткое описание

- Разработанное приложение состоит из двух частей: серверной и клиентской.
- Серверная часть реализована с использованием **FastAPI** на Python и обеспечивает корректную передачу данных из базы данных **Neo4j** на клиентскую сторону. Для взаимодействия между сервером и клиентом используется **RESTful API**, что позволяет эффективно передавать данные для отображения в интерфейсе.
- Клиентская часть приложения получает данные через API и отображает их в удобочитаемом виде. Для разработки использовалась **модульная архитектура**, где проект разделен на папки для компонентов, страниц, виджетов и общих ресурсов. В проекте применяется **TypeScript** для строгой типизации, что улучшает поддержку и масштабируемость кода.
- Для развертывания и обслуживания приложения используется **Docker**.

4.2 Используемые технологии:

- **База данных:** MongoDB
- **Backend:** RESTful API, Python, FastAPI, neo4j
- **Frontend:** TypeScript
- **Развертывание:** Docker

4.3 Схема экранов приложения

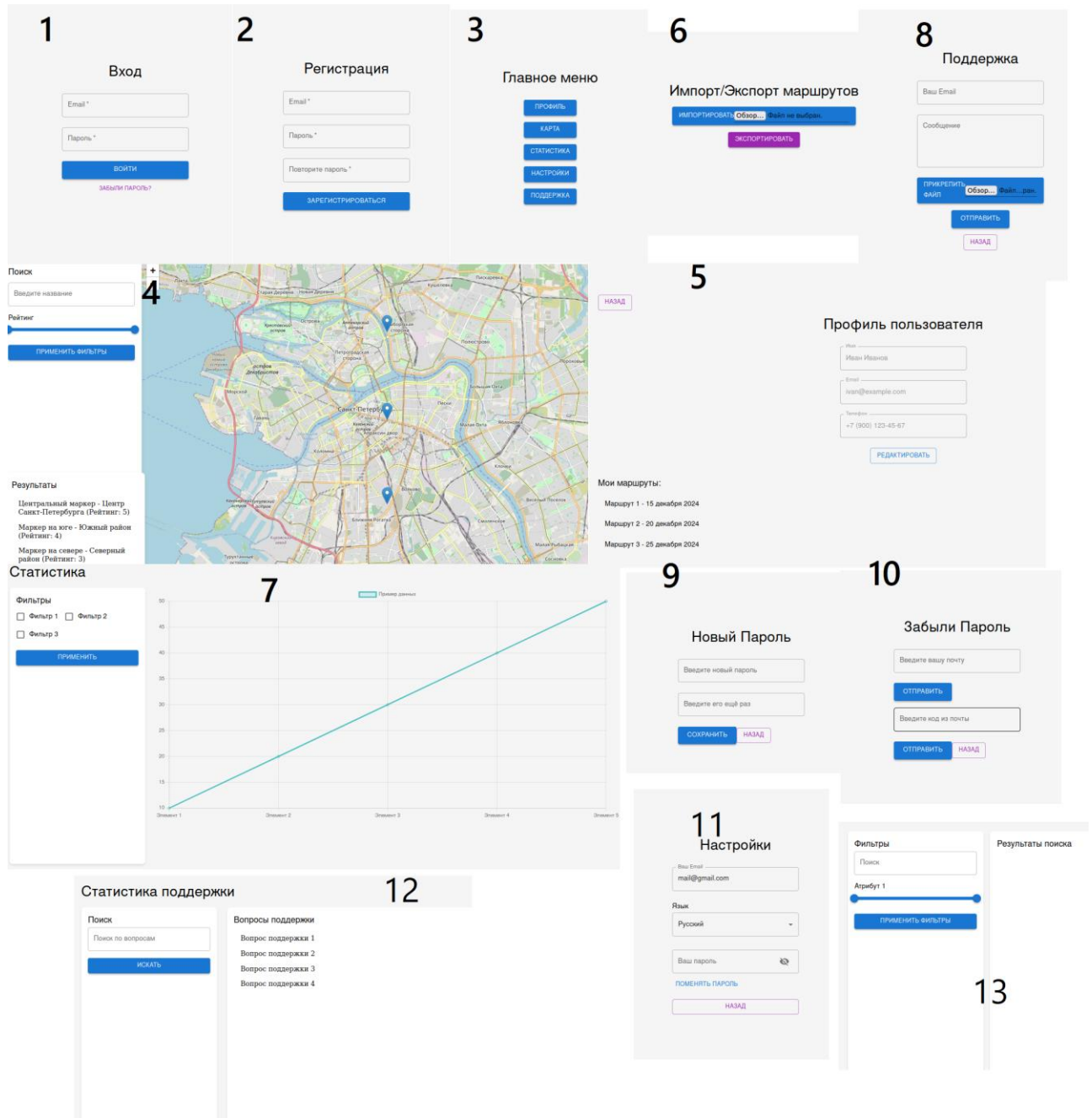


Рисунок 9. Все экраны

5. ВЫВОДЫ

5.1 Достигнутые результаты В ходе работы над проектом была разработана платформа для управления доступностью маршрутов, которая предоставляет пользователям возможности оценивать и планировать маршруты с учетом доступности объектов. Для пользователей создан удобный интерфейс для поиска и сохранения маршрутов, а также просмотра и фильтрации по различным параметрам доступности.

Для администраторов реализованы функции управления данными о точках доступности, импорта и экспорта данных, что улучшает процесс управления контентом и облегчает миграцию данных. Также администраторы могут анализировать статистику использования платформы для оптимизации и планирования развития ресурсов.

5.2 Недостатки и пути для улучшения В текущей реализации отсутствует поддержка интерактивных карт и навигации в реальном времени, что могло бы значительно улучшить пользовательский опыт. Также ограничен функционал по настройке уведомлений о изменении условий доступности маршрутов.

Платформа доступна только на одном языке, что ограничивает её использование в международном контексте. Расширение поддержки языков может значительно увеличить аудиторию пользователей.

На данный момент платформа не оптимизирована для мобильных устройств, что снижает ее доступность и удобство использования на смартфонах и планшетах.

5.3 Будущее развитие решения в будущем планируется внедрение функционала интерактивной карты и навигации в реальном времени для обеспечения более глубокого взаимодействия с пользователями. Расширение языковой поддержки и адаптация интерфейса для мобильных устройств позволит охватить более широкую аудиторию и улучшить пользовательский опыт. Также предусмотрено внедрение системы уведомлений, которая будет информировать пользователей о важных изменениях в доступности маршрутов.

6. ПРИЛОЖЕНИЯ

- **6.1 Документация по сборке и разворачиванию приложения.**
- 1. Склонировать репозиторий проекта с помощью команды `git clone https://github.com/moevm/nosql2h24-water.git`.
- 2. Выполнить сборку образов: `docker-compose build`.
- 3. Произвести запуск контейнеров командой: `docker-compose up`.
- Клиентская часть приложения доступна в браузере по адресу `http://127.0.0.1:8080`.
- **6.2 Инструкция для пользователя.**
- Требования:
- В системе должна быть доступна платформа контейнеризации Docker и быть свободными порты 8080 и 8000.
- Возможности:
- - просмотр карты и меток;
- - просмотр и редактирование информации своего профиля;
- - отправка запросов поддержки;
- - просмотр метрик сайта;
- - настройка интерфейса пользователя;

7. ЛИТЕРАТУРА

1. Репозиторий проекта - [Электронный ресурс].-URL:
<https://github.com/moevm/nosql2h24-water/>