

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра №806 «Вычислительная математика и программирование»

**Итоговый проект
по курсу «Базы данных»**

Сервис по передержке котов

Выполнила: Старостина А. А.

Группа: М8О-308Б-22

Преподаватель: Малахов А. В.

Москва, 2024

Оглавление

1. Постановка задачи
2. Описание
3. Модели предметной области
4. Модели уровня инфраструктуры
5. Запросы к базе данных
6. Графический интерфейс
7. Вывод

Постановка задачи

Общие требования:

1. При реализации курсового проекта должно быть использование СУБД PostgreSQL.
2. Необходимо выбрать предметную область для создания базы данных. Выбранная предметная область должна быть уникальной для всего потока, а не только в рамках учебной группы.
3. Необходимо описать модели предметной области и уровня инфраструктуры и их назначение в рамках реализуемого проекта (минимальное количество моделей предметной области и уровня инфраструктуры - 5). Также необходимо выполнить проектирование логической структуры базы данных. Все таблицы, связанные с описанными моделями предметной области, должны находиться в 3NF или выше. База данных должна иметь минимум 7 таблиц.
4. Клиентское приложение должно быть в виде WEB или оконного приложения.
5. Необходимо организовать различные роли пользователей и права доступа к данным (например: администратор, редактор, рядовой пользователь). Клиентское приложение, взаимодействующее с базой данных, должно предоставлять функционал для авторизации пользователя по логину и паролю (хранение непосредственно пароля в базе данных запрещено, надо хранить HASH от этого пароля и проверять его).
6. При разработке функционала базы данных следует организовать логику обработки данных не на стороне клиента (Frontend), а на стороне серверного приложения (Backend). Все обработки «SQL запросов», «работа бизнес-логики должны находиться в BACKEND части. FRONT только для отображения.
7. Запросы должны быть асинхронны. То есть, при нажатии на форму она не должна зависать. При нажатии на форму одним пользователем, другой должен иметь возможность свободно пользоваться приложением. То есть действия разных пользователей независимы.

8. Необходимо реализовать возможность создания администратором архивных копий базы данных и восстановления данных из клиентского приложения.

9. Передача параметров в SQL запрос должна происходить только через parameters.

Описание

Проект представляет собой веб-сервис для передержки котов, который позволяет пользователям искать доступные адреса, бронировать места и оплачивать их. Пользователи могут зарегистрироваться в системе, авторизоваться и управлять своими бронированиями через понятный интерфейс. Система обеспечивает создание и управление бронированиями, проверку доступных мест на адресах, а также обработку платежей. Администраторы могут управлять базой данных, создавать и удалять адреса, создавать резервные копии для обеспечения надежности работы. Вся информация о пользователях, котах, местах, бронированиях и платежах хранится в базе данных PostgreSQL. Веб-интерфейс для работы с сервисами реализован с использованием Streamlit. Для взаимодействия с базой данных используется psycopg2.

Проект включает в себя сервисы для авторизации и регистрации пользователей, обработки бронирований, управления платежами, а также сервис для администрирования базы данных.

Модели предметной области

Модель представляет структуру для хранения информации о пользователях, котах, бронированиях, местах и платежах. Эта система помогает пользователям дать на хранение проверенным людям своего любимого животного (и даже не одного) в любом удобном для себя месте при наличии свободных мест. Администратор может связаться с хозяином животного по почте. Модель включает в себя 5 таблиц:

1. users – информация о пользователях (и клиенты, и администраторы)
2. places – информация о местах (адресах)
3. cats – информация о котах (кошках)
4. bookings – информация о бронированиях
5. payments – информация о платежах

users (Пользователи):

user_id - 'Уникальный идентификатор пользователя';
first_name - 'Имя пользователя';
last_name - 'Фамилия пользователя';
email - 'Email пользователя';
password_hash - 'Хеш пароля пользователя';
role - 'Роль пользователя ("admin", "user")';

Связана с таблицей bookings для отслеживания бронирований (один ко многим). То есть один пользовател может сделать несколько бронирований.

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    role VARCHAR(50) NOT NULL CHECK (role IN ('user', 'admin'))
);
```

places (Места):

place_id - 'Уникальный идентификатор места';
address - 'Адрес места';
count_free - 'Количество свободных мест';

Связана с таблицей bookings для отслеживания бронирований (один ко многим). То есть у одного места(адреса) может быть несколько бронирований.

```
CREATE TABLE places (
    place_id SERIAL PRIMARY KEY,
    address VARCHAR(100) NOT NULL,
    count_free INT NOT NULL CHECK (count_free BETWEEN 0 AND 5)
);
```

cats (коты):

cat_id - 'Уникальный идентификатор кота';
nickname - 'Кличка кота';
age - 'Возраст кота';
features - 'Особенности кота';
breed - 'Порода кота';

Связана с таблицей bookings для отслеживания бронирований (один ко многим). То есть один кот может быть забронирован несколько раз.

```
CREATE TABLE cats (
    cat_id SERIAL PRIMARY KEY,
    nickname VARCHAR(100) NOT NULL,
    age INT NOT NULL,
    features VARCHAR(100) NOT NULL,
    breed VARCHAR(50) NOT NULL
);
```

bookings (бронирования):

booking_id - 'Уникальный идентификатор бронирования';
user_id - 'Идентификатор пользователя, который сделал бронирование';
cat_id - 'Идентификатор кота, для которого было сделано бронирование';
place_id - 'Идентификатор места, в котором было забронировано';
booking_date - 'Дата бронирования';

Связана с таблицами places, users, cats (многое к одному), payments(один ко многим).

```
CREATE TABLE bookings (
    booking_id SERIAL PRIMARY KEY,
    user_id INT NOT NULL,
    cat_id INT NOT NULL,
    place_id INT NOT NULL,
    booking_date DATE NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (cat_id) REFERENCES cats(cat_id),
    FOREIGN KEY (place_id) REFERENCES places(place_id)
);
```

payments (платежи):

payment_id - 'Уникальный идентификатор платежа';
booking_id - 'Идентификатор бронирования, к которому привязан платеж';
payment_date - 'Дата платежа';
amount - 'Сумма платежа';
status - 'Статус платежа';
card_number - 'Номер карты';
card_expiry - 'Срок действия карты';
card_cvc - 'CVC';

Связана с таблицей bookings для отслеживания бронирований (многое к одному). То есть может быть много платежей у одного бронирования (на одно и то же место, на тот же адрес).

```
CREATE TABLE payments (
    payment_id SERIAL PRIMARY KEY,
    booking_id INT NOT NULL,
    payment_date DATE NOT NULL,
    amount DECIMAL(10, 2) NOT NULL CHECK (amount >= 0),
    status VARCHAR(50) NOT NULL,
    card_number VARCHAR(20) NOT NULL,
    card_expiry VARCHAR(7) NOT NULL,
    card_cvc VARCHAR(4) NOT NULL,
    FOREIGN KEY (booking_id) REFERENCES bookings(booking_id)
);
```

Триггер bookings_after_insert

Выполняется после добавления записи в bookings, он уменьшает count_free на 1, чтобы обеспечивать своевременный доступ к свободным местам.

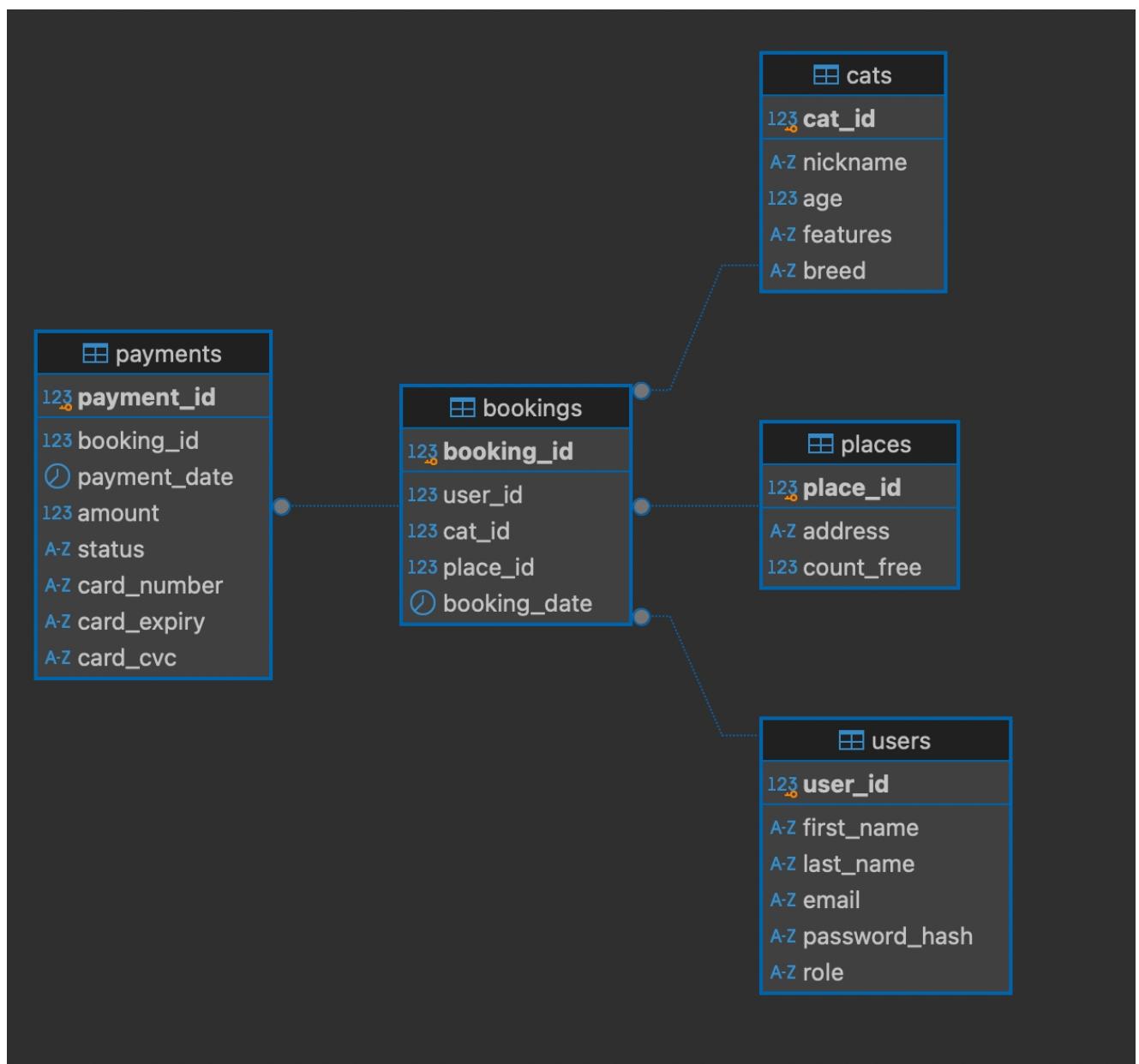
```
-- Функция триггера, уменьшающая count_free для места
CREATE OR REPLACE FUNCTION decrease_place_count_free()
RETURNS TRIGGER AS $$

BEGIN
    -- Уменьшаем count_free на 1 для места из новой записи в bookings
    UPDATE places
    SET count_free = count_free - 1
    WHERE place_id = NEW.place_id;
    RETURN NEW;
END;

$$ LANGUAGE plpgsql;
```

```
-- Триггер, выполняющийся после добавления записи в bookings
CREATE TRIGGER bookings_after_insert
AFTER INSERT ON bookings
FOR EACH ROW
EXECUTE FUNCTION decrease_place_count_free();
```

Общая схема моделей предметной области



Модели уровня инфраструктуры

Сервис авторизации/регистрации:

Позволяет создавать новые аккаунты и заходить в уже существующие. Пароль хранятся в захешированном виде, что обеспечивает безопасность.

Основные функции:

- Регистрация новых пользователей
- Авторизация (с проверкой email)
- Обработка паролей
- Управление ролями (admin, user)

Сервис бронирования:

Позволяет обрабатывать операции, связанные с поиском подходящего бронирования: учитывает дату, адрес, количество свободных мест.

Основные функции:

- Получение списка мест с количеством свободных мест на указанную дату
- Подробная информация о бронировании определенного адреса
- Создание бронирования
- Добавление нового адреса
- Удаление старого адреса (с проверкой, что нет запланированных бронирований в будущем)

Сервис платежей:

Позволяет обрабатывать платежи, обновлять их статус.

Основные функции:

- Обработка платежей после начала бронирования
- Обновление статуса «Оплачено/Не оплачено»
- Хранение платежей

Сервис администрирования БД:

Следит за безопасностью и сохранностью данных.

Основные функции:

- Создание резервной копии
- Восстановление данных по резервной копии

Запросы к базе данных

```
def book_place(self, user_id: int, cat_id: int, place_id: int, booking_date: str) -> Optional[int]:
    with get_connection() as conn:
        with conn.cursor() as cur:
            try:
                conn.autocommit = False
                cur.execute("""
                    INSERT INTO bookings (user_id, cat_id, place_id, booking_date)
                    VALUES (%s, %s, %s, %s)
                    RETURNING booking_id;
                """, (user_id, cat_id, place_id, booking_date))
                booking_id = cur.fetchone()[0]
                conn.commit()
                return booking_id
            except Exception as e:
                conn.rollback()
                raise e
```

```
def create_cat(self, nickname: str, age: int, features: str, breed: str) -> Optional[int]:
    with get_connection() as conn:
        with conn.cursor() as cur:
            try:
                cur.execute(
                    """
                    INSERT INTO cats (nickname, age, features, breed)
                    VALUES (%s, %s, %s, %s)
                    RETURNING cat_id;
                """
                , (nickname, age, features, breed))
            )
            cat_id = cur.fetchone()[0]
            conn.commit()
            return cat_id
        except Exception as e:
            conn.rollback()
            raise e
```

```
def get_bookings_by_place(self, place_id: int, date: str) -> List[Dict[str, Optional[int]]]:
    query = """
        SELECT b.booking_id, u.first_name, u.email, c.nickname, b.booking_date
        FROM bookings b
        JOIN users u ON b.user_id = u.user_id
        JOIN cats c ON b.cat_id = c.cat_id
        WHERE b.place_id = %s AND b.booking_date = %s;
    """
    with get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute(query, (place_id, date,))
            return [
                {
                    "booking_id": row[0],
                    "user_first_name": row[1],
                    "email": row[2],
                    "cat_nickname": row[3],
                    "booking_date": row[4]
                } for row in cur.fetchall()
            ]
```

```
def get_user_bookings(self, user_id: int) -> List[Dict[str, Optional[int]]]:  
    """ Получает все бронирования для указанного пользователя. """  
    query = """  
        SELECT b.booking_id, p.address, b.booking_date, c.nickname  
        FROM bookings b  
        JOIN places p ON b.place_id = p.place_id  
        JOIN cats c ON b.cat_id = c.cat_id  
        WHERE b.user_id = %s  
        ORDER BY b.booking_date ASC;  
    """  
  
    with get_connection() as conn:  
        with conn.cursor() as cur:  
            cur.execute(query, (user_id,))  
            return [{  
                "booking_id": row[0],  
                "place_address": row[1],  
                "booking_date": row[2],  
                "catNickname": row[3]  
            } for row in cur.fetchall()]
```

```
def get_booking_info(self, booking_id: int) -> Optional[Dict[str, Optional[int]]]:  
    """ Получает информацию о конкретном бронировании. """  
    query = """  
        SELECT b.booking_id, u.first_name, u.last_name, c.nickname, p.address, b.booking_date  
        FROM bookings b  
        JOIN users u ON b.user_id = u.user_id  
        JOIN cats c ON b.cat_id = c.cat_id  
        JOIN places p ON b.place_id = p.place_id  
        WHERE b.booking_id = %s;  
    """  
  
    with get_connection() as conn:  
        with conn.cursor() as cur:  
            cur.execute(query, (booking_id,))  
            row = cur.fetchone()  
            if row:  
                return {  
                    "booking_id": row[0],  
                    "user_first_name": row[1],  
                    "user_last_name": row[2],  
                    "catNickname": row[3],  
                    "place_address": row[4],  
                    "booking_date": row[5]  
                }  
            else:  
                print(f"No booking found for booking id {booking_id}")  
                return None
```

```
def create_payment(self, booking_id : int, amount : int, card_number : int, card_expiry : int, card_cvc : int) :
    query = """
        INSERT INTO payments (booking_id, payment_date, amount, status, card_number, card_expiry, card_cvc)
        VALUES (%s, %s, %s, %s, %s, %s, %s)
        RETURNING payment_id;
    """

    with get_connection() as conn:
        with conn.cursor() as cur:
            payment_date = datetime.now().date()
            cur.execute(query, (
                booking_id,
                payment_date,
                amount,
                "Не оплачено",
                card_number,
                card_expiry,
                card_cvc
            ))
            payment_id = cur.fetchone()[0]
            conn.commit()
    return payment_id
```

```
def get_user_by_email(self, email: str):
    query = """
        SELECT * FROM users WHERE email = %(email)s;
    """

    with get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute(query, {
                "email": email
            })
    return cur.fetchone()
```

```
def add_user(self, first_name: str, last_name: str, email: str, password_hash: str, role: str = "user"):
    query = """
        INSERT INTO users (first_name, last_name, email, password_hash, role)
        VALUES (%(first_name)s, %(last_name)s, %(email)s, %(password_hash)s, %(role)s);
    """

    with get_connection() as conn:
        with conn.cursor() as cur:
            cur.execute(query, {
                "first_name": first_name,
                "last_name": last_name,
                "email": email,
                "password_hash": password_hash,
                "role": role
            })
            conn.commit()
```

Графический интерфейс
(streamlit run src/main.py)

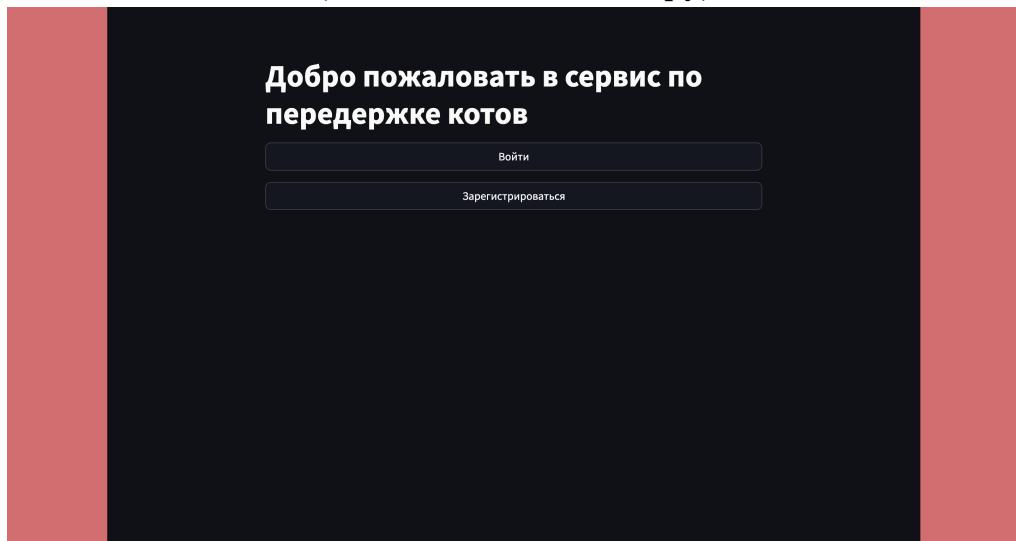


Рис. 1

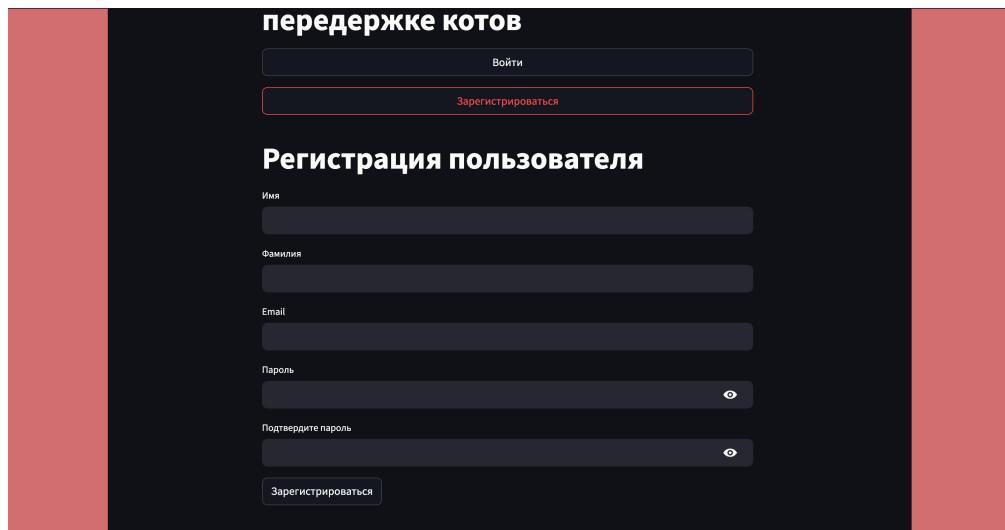


Рис. 2

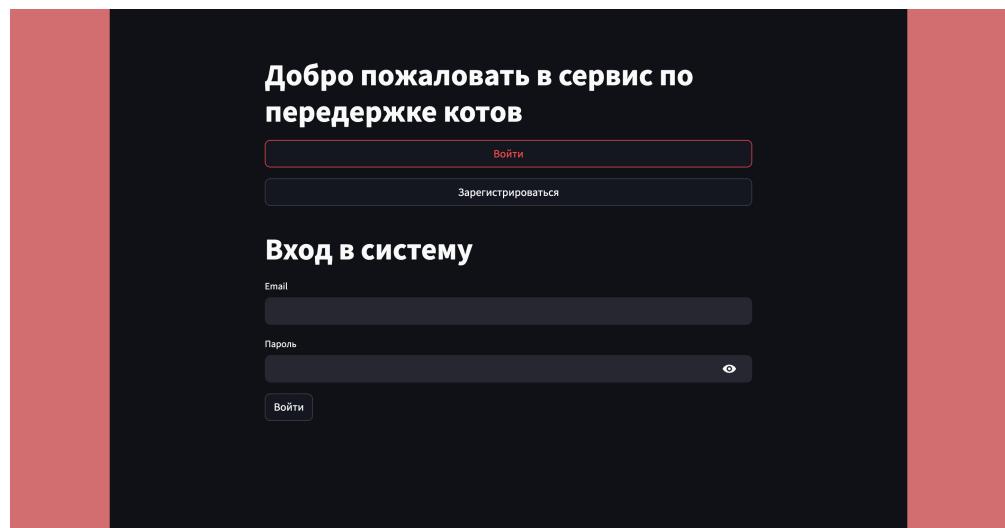


Рис. 3

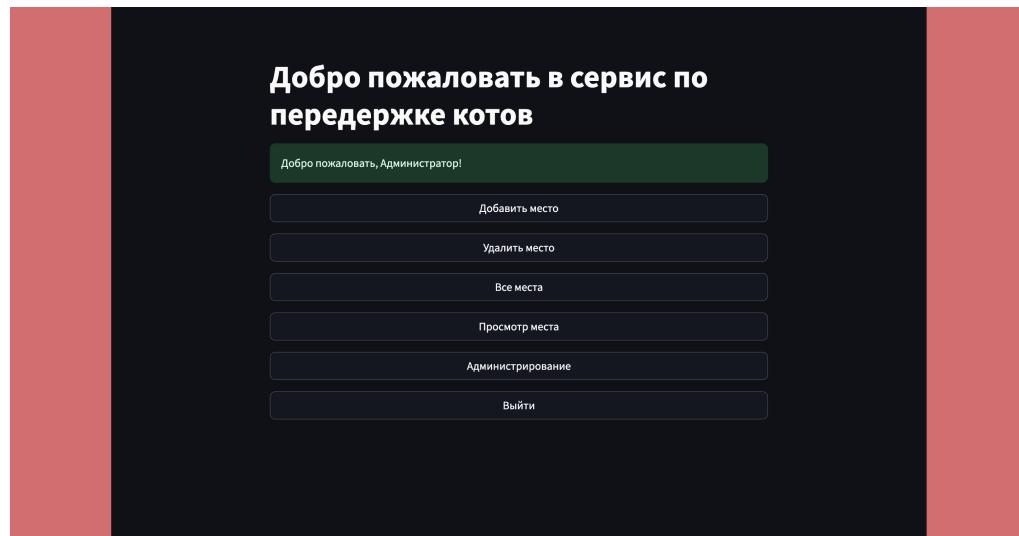


Рис. 4

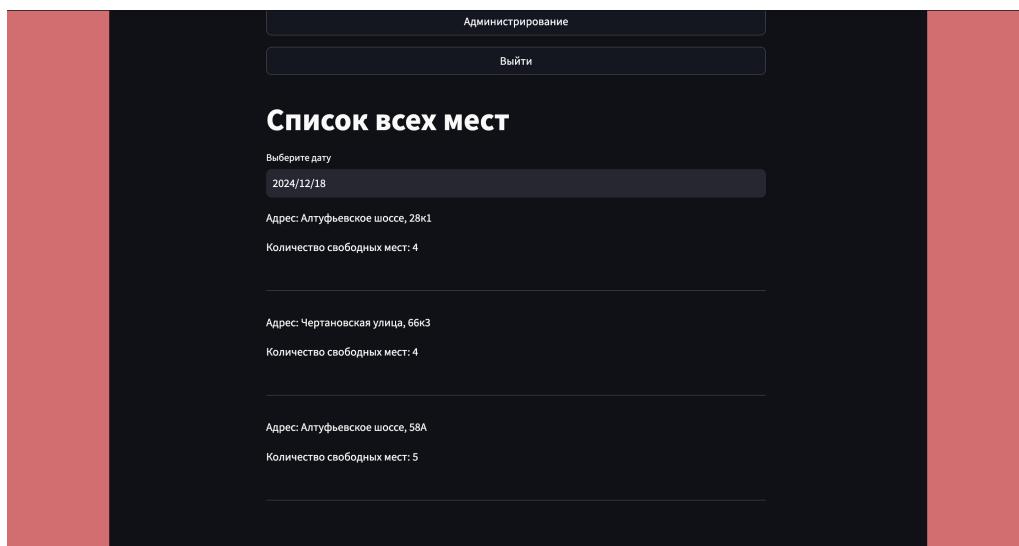


Рис. 5

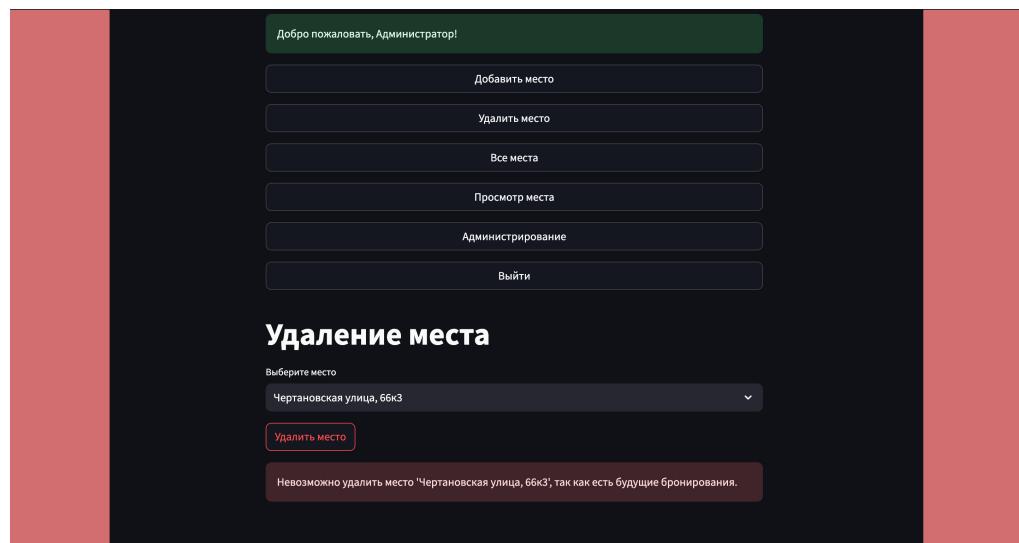


Рис. 6

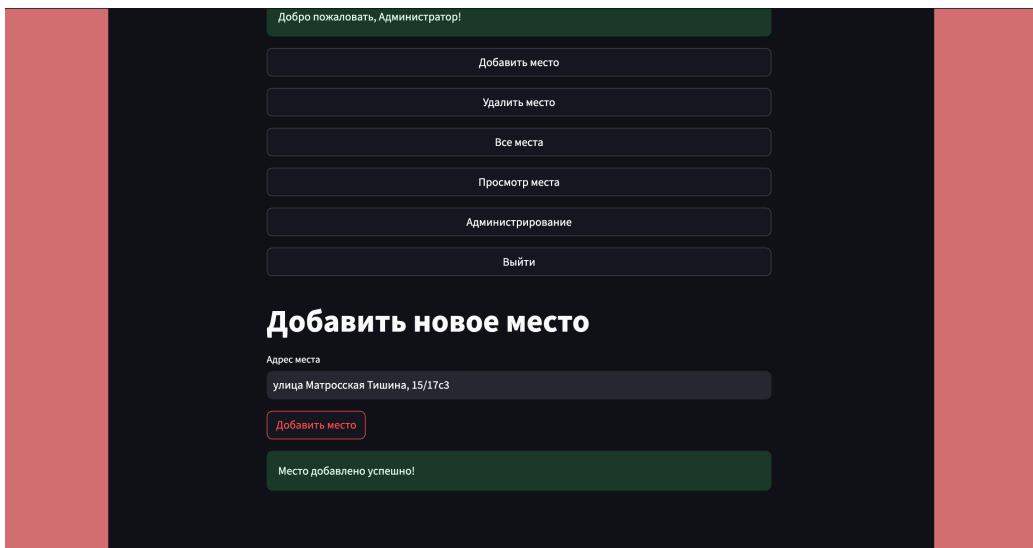


Рис. 7

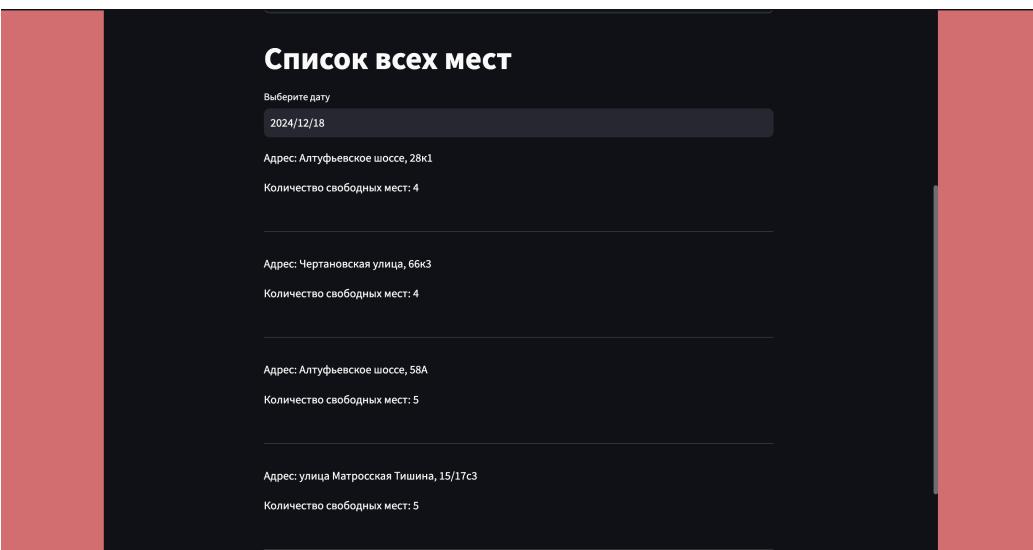


Рис. 8

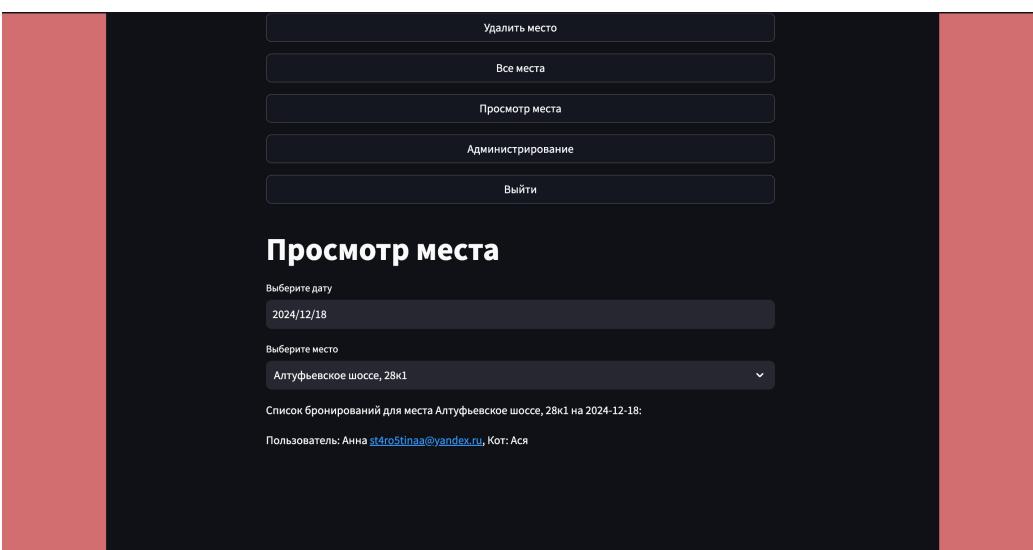


Рис. 9

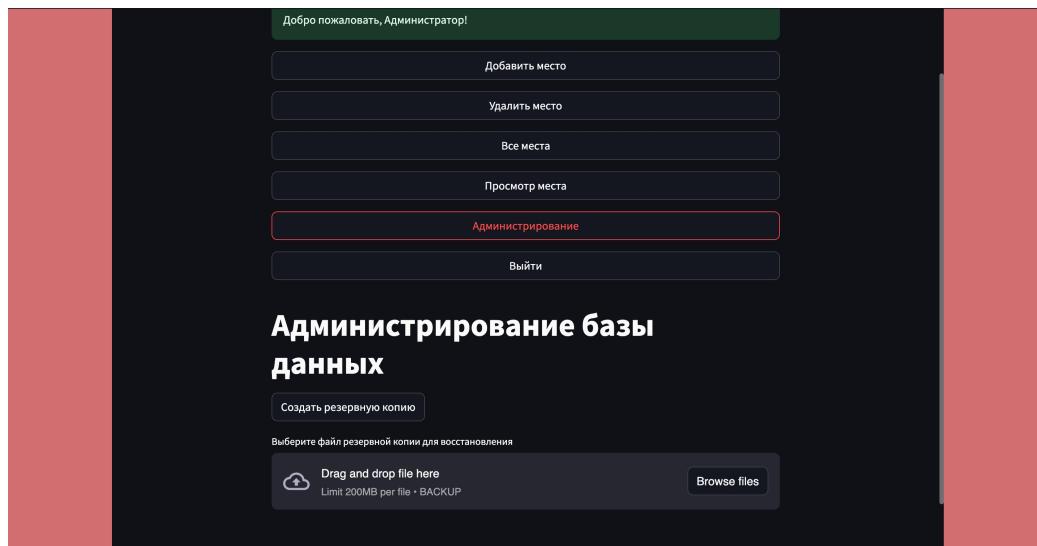


Рис. 10

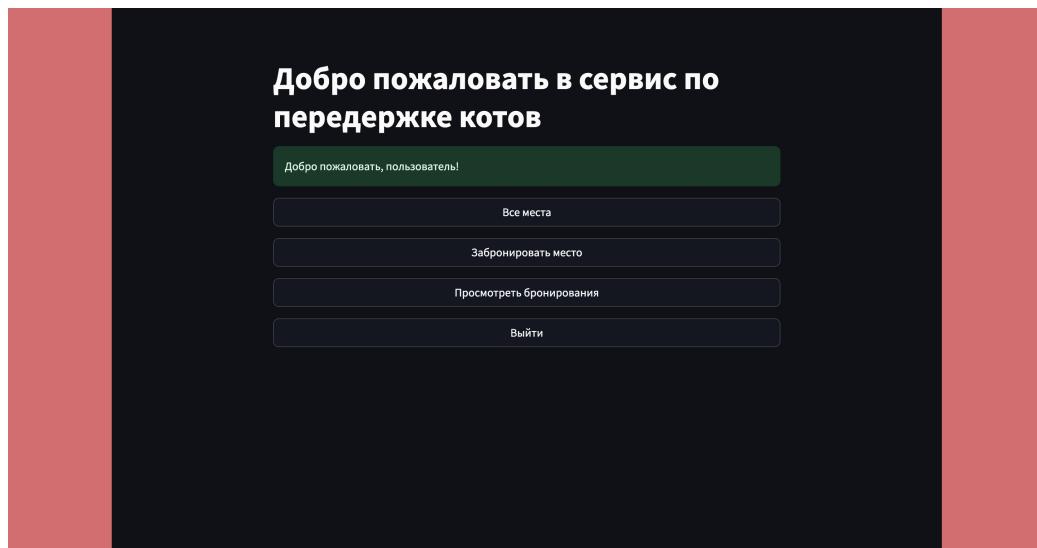


Рис. 11

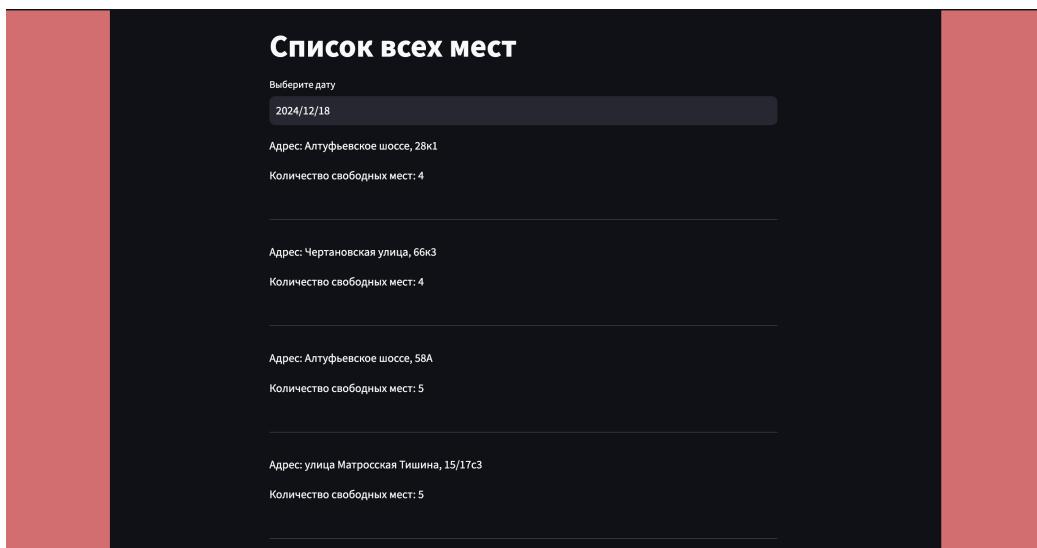


Рис. 12

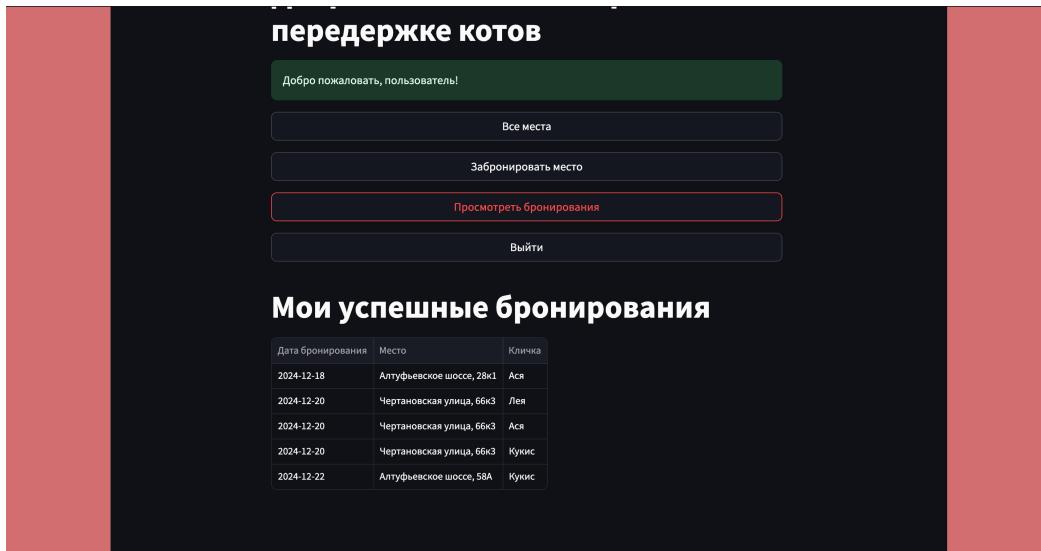


Рис. 13

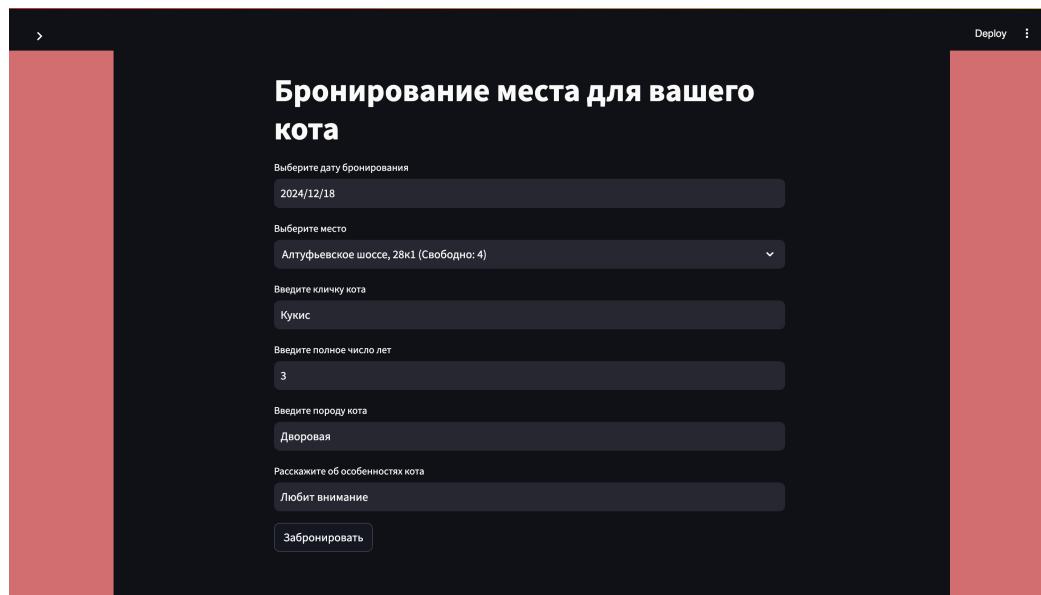


Рис. 14

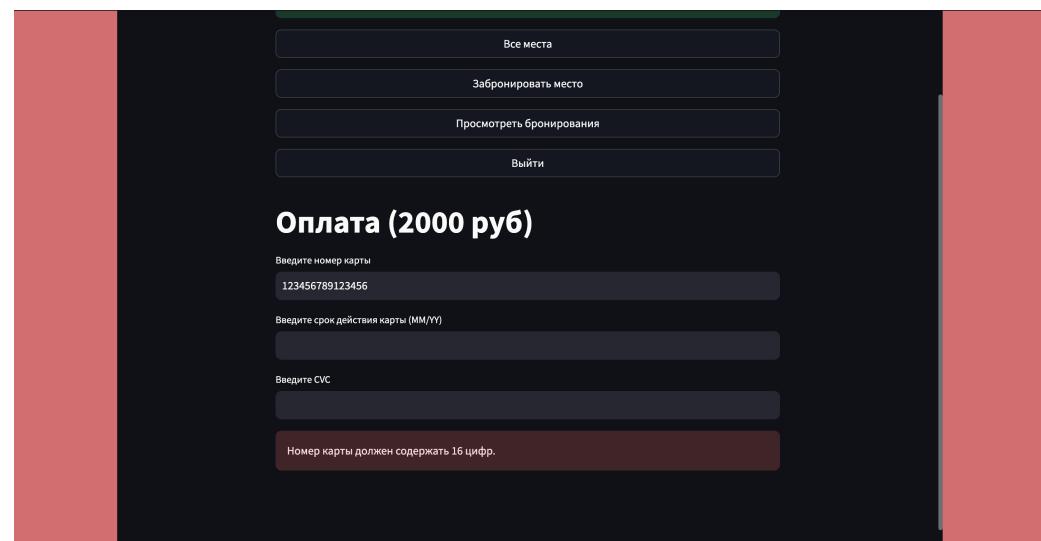


Рис. 15

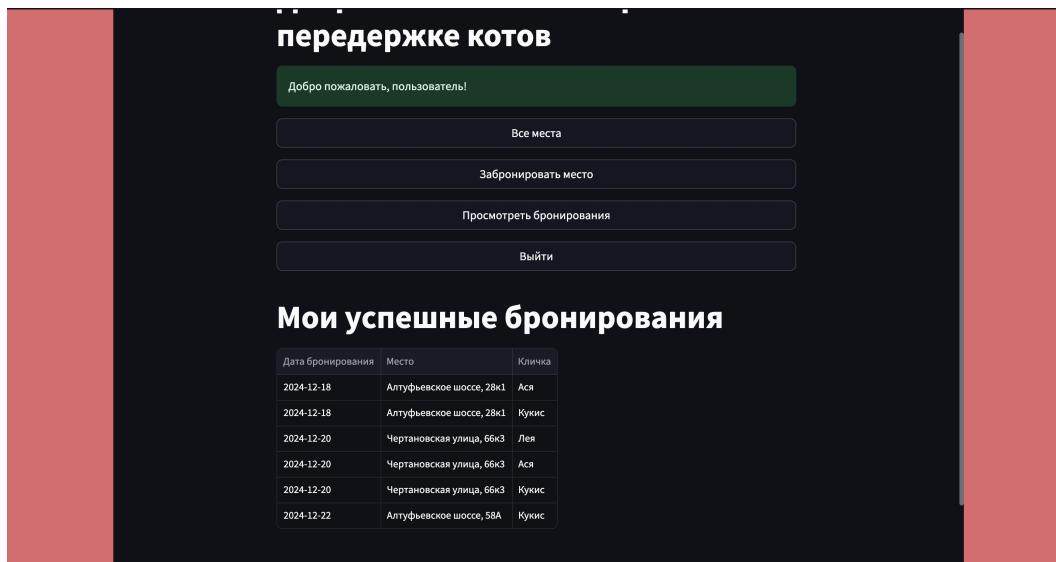


Рис. 16

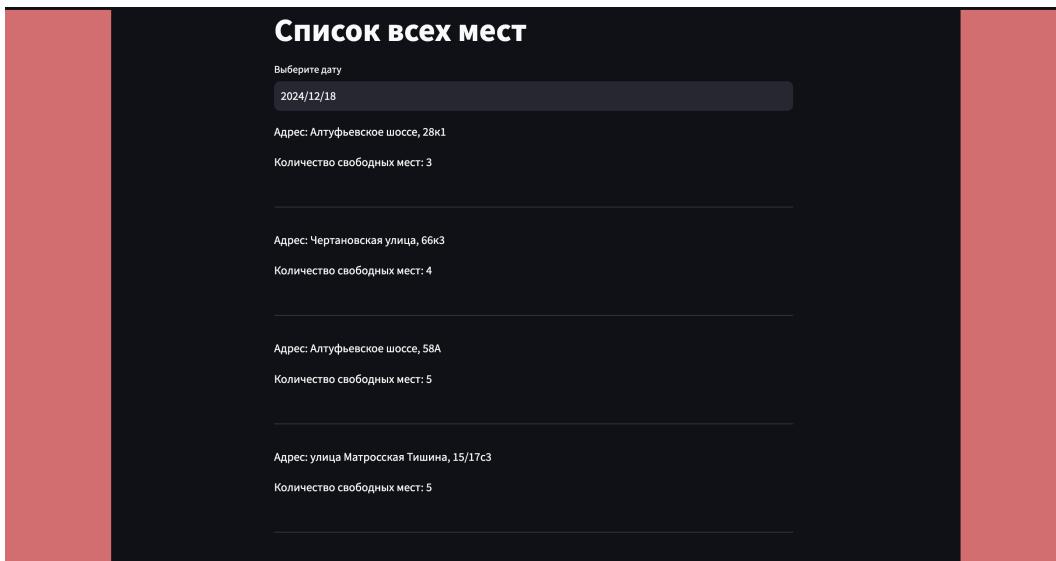


Рис. 17

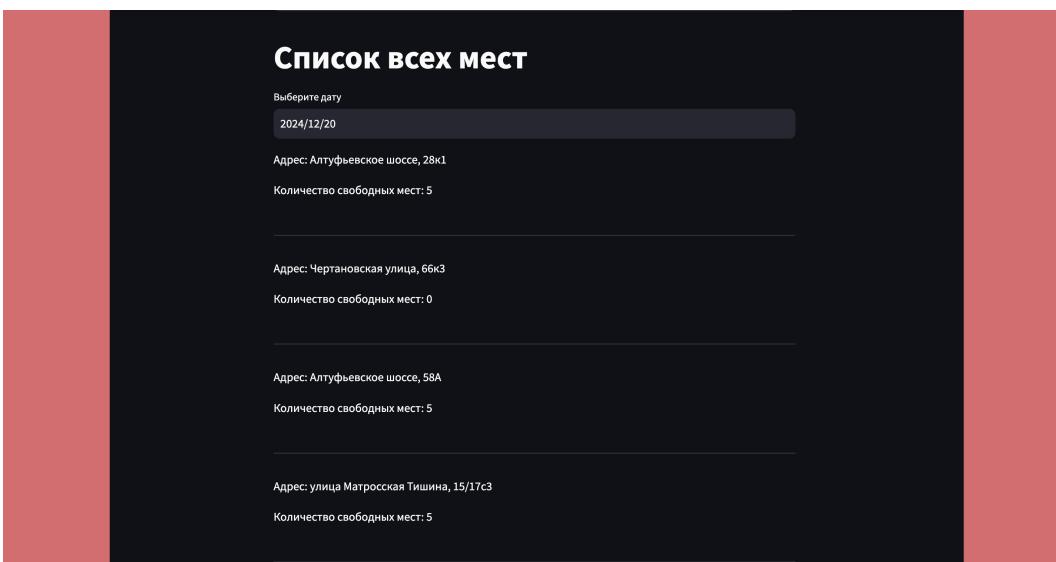


Рис. 18

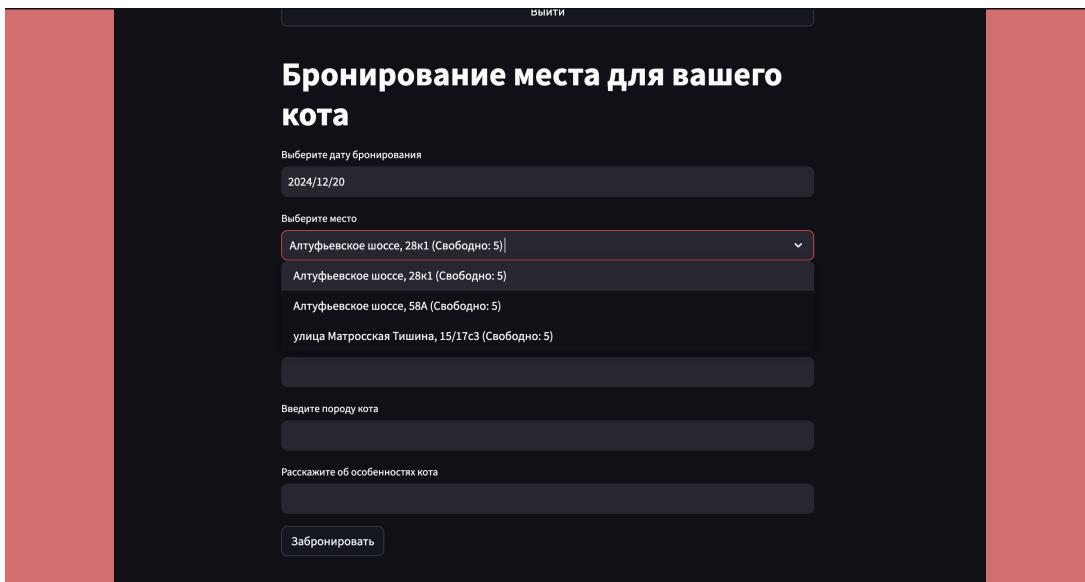


Рис. 19

Описание:

- На рисунке 1 пользователя встречает приветствие и выбор: зайти/зарегистрироваться.
- На рисунке 2 можно понять, какие данные требуются для успешной регистрации.
- На рисунке 3 планируется зайти в систему как администратор.
- На рисунке 4 мы видим главные функции для администратора.
- На рисунке 5 мы можем просмотреть список всех мест, предварительно выбрав нужную дату. В списке будет показываться количество свободных мест на эту дату (остальное забронировано. максимум: 5). Сейчас видно 3 адреса.
- На рисунке 6 администратор хочет удалить адрес, но так как в дальнейшем запланированы передержки, операция не будет выполнена.
- На рисунке 7 администратор добавил новое место.
- На рисунке 8 администратор снова смотрит список всех адресов и видит, что новое место действительно добавилось.
- На рисунке 9 администратор может просмотреть отдельное место(адрес) на определенную дату, может видеть дополнительную информацию, которую не видит обычный пользователь: имя пользователя, его email, кличка его кота.
- На рисунке 10 можно совершить резервную копию (сначала создать копию, затем с помощью команд из readme в моем проекте снести бд, затем восстановить)
- На рисунке 11 заходит пользователь.
- На рисунке 12 он видит список адресов точно так же, как и администратор.
- На рисунке 13 пользователь может посмотреть все его успешные (оплаченные) бронирования за все время.

- На рисунке 14 пользователь бронирует место для своего кота.
- На рисунке 15 пользователь хочет оплатить, но он ввел не 16 цифр для карты.
- На рисунке 16 пользователь успешно оплатил бронирование и оно у него появилось в его списке.
- На рисунке 17 заметно, что количество мест на день и адрес, который выбрал пользователь, уменьшилось на 1.
- На рисунке 18 пользователь смотрит на дату 2024/12/20 и видит, что по адресу "Чертановская 6бк3" не осталось мест.
- На рисунке 19 пользователь хочет все равно выбрать этот день и этот адрес, но его нет в списке (это обрабатывается).

Выводы

Благодаря проекту был самостоятельно реализован сервис для помощи людей с любимыми питомцами: сервис по передержке котов. Это франшиза распространена по городу Москва, поэтому адреса касаются только этого города. Если им нужно куда-то уехать, они могут воспользоваться данным сервисом.

У администратора довольно важные функции: он может создавать, удалять адреса, делать резервную копию, следить за бронированиями (кто кого когда забронировал).

Пользователь же может смотреть весь список адресов на определенную дату и выбирать то, что ему больше всего подходит, а также оплачивать бронирования.

Проект позволил глубже познакомиться с устройством PostgreSQL, что очень может помочь в будущей практике.