ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина «Теория баз данных»

Отчет о проделанной работе:
«Разработка чат-бота в Telegram с поддержкой базы данных»

Выполнили: студенты
Смирнова Анна Романовна БПМИ194
Григорьева Анастасия Юрьевна БПМИ 199
Вьялков Василий Владимирович БПМИ197

Преподаватель:
Незнанов Алексей Андреевич

_____ _____

оценка *подпись*

Москва 2021

Оглавление

Описание предметной области

Целью работы является разработка программной системы, которая хранит данные о доступных в прокате фильмах, кинотеатрах, где эти фильмы показываются, сеансах и свободных местах. Также система предоставляет возможность бронирования билета на определенный сеанс. Данные хранятся в серверной части проекта (на Python и SQL), работа с базой данных происходит через клиентскую часть (приложение чат-бот в Telegram).

Основной сценарий использования

Используя чат-бот, пользователь может найти подходящий кинотеатр или фильм, подобрать сеанс, выбрать место в зале и забронировать билет. Кроме этого, доступна возможность посмотреть топ самых популярных фильмов за последнее время.

Данные

Большая часть данных была сгенерирована вручную, однако для проведения небольшого анализа в конце работы были заимствованы данные из открытого источника – Kaggle, система для организации конкурсов по исследованию данных, где хранится множество подходящих датасетов.

Концептуальное проектирование

Концептуальная модель

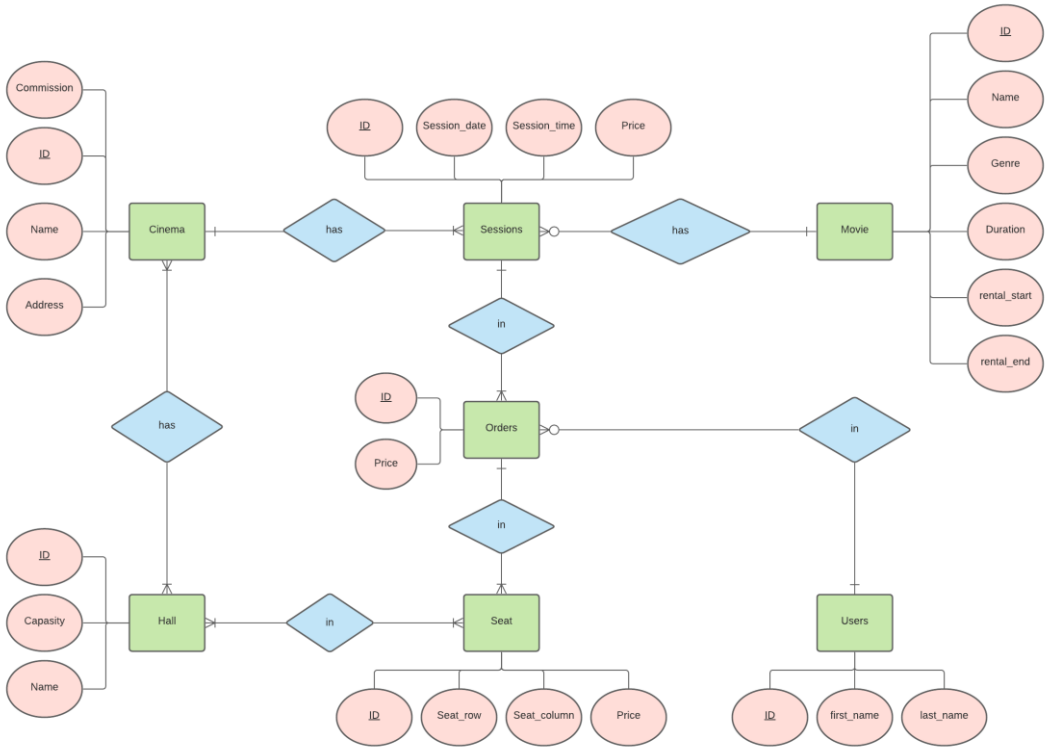


Рисунок 1 – Концептуальная модель в нотации Crow's-foot

В процессе разработки было выделено 7 основных сущностей. Все они являются минимальным набором для корректной работы бота и уникальны. Кроме того, в процессе дальнейшей разработки бота есть возможность переводить в сущности некоторые атрибуты и менять структуру.

Разберемся, какой получился словарь сущностей с описанием каждой:

- Пользователь (user) - содержит информацию о пользователе (атрибуты: ID, имя, фамилия)
- Фильм (movie) - содержит информацию о фильме (атрибуты: ID, название, жанр, длительность, начало проката, конец проката)
- Сеанс (session) - содержит информацию о киносеансе (атрибуты: ID, дата, время, цена)
- Кинотеатр (cinema) - место, в котором проводятся киносеансы (атрибуты: ID, комиссия кинотеатра, название, адрес)
- Заказ (order) - содержит информацию о покупке (атрибуты: ID, цена)
- Зал (hall) – место показа кино (атрибуты: ID, вместимость, номер зала)
- Место (seat) – место посадки пользователя (атрибуты: ID, номер ряда, номер места, цена)

ER-диаграмма

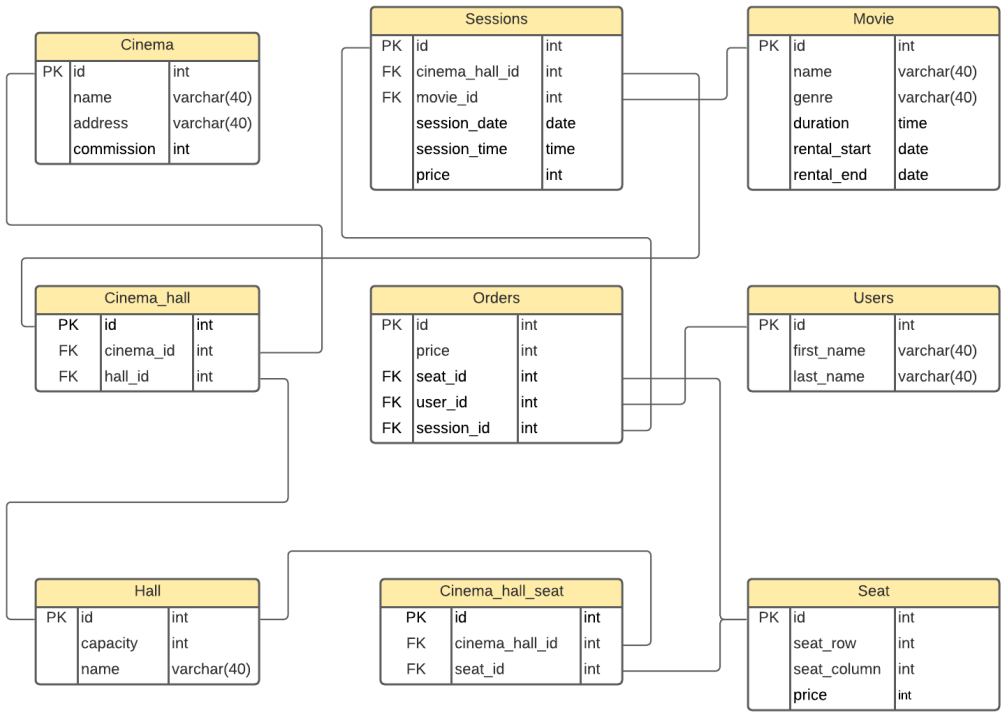


Рисунок 2 – Логическая модель

При построении реляционной модели были установлены внешние ключи (FK) для всех связей. Связи [многие ко многим] были организованы в виде таблиц пар ключей: «Cinema_hall», которая отражает связь кинотеатров и залов, и «hall_seat», которая связывает залы и занятые места. При переходе от концептуальной модели к логической были несколько раз изменены структуры, для удобства работы с базой данных и разработкой бота. В отчете представлена только финальная версия обеих моделей.

Развёртывание БД и написание бота

Используемые средства разработки

При разворачивании базы данных нами использовалась среда DataGrip 2021.2.2**,** версия PostgreSQL 14.1. База данных разворачивалась на учебном сервисе postgres@localhost. Бот был разработан на Python последней версии, с помощью библиотек aiogram и asyncpg - это асинхронная библиотека для работы с базами данных с PostgreSQL.

DDL-скрипт создания схемы БД

```
create table nav_movie.users
    (
     id int primary key,
     first_name varchar(40),
     last_name varchar(40)
    );

create table nav_movie.movie
    (
        id int primary key,
        name varchar(40),
        genre varchar(40),
        duration time,
        rental_start date,
        rental_end date
    );

create table nav_movie.cinema
    (
        id int primary key,
        name varchar(40),
        address varchar(40)
    );

create table nav_movie.hall
    (
        id int primary key,
        capacity int,
        name varchar(40)
    );
```

```sql
create table nav_movie.sessions
    (
        id int primary key,
        cinema_hall_id int check(cinema_hall_id>0) references nav_movie.cinema_hall(id),
        movie_id int check(movie_id>0) references nav_movie.movie(id),
        session_date date,
        session_time time
    );

create table nav_movie.cinema_hall
    (
        id int primary key,
        cinema_id int check(cinema_id>0) references nav_movie.cinema(id),
        hall_id int check (hall_id>0) references nav_movie.hall(id)
    );

create table nav_movie.seat
    (
        id int primary key,
        seat_row int check (seat_row>0),
        seat_column int check (seat_column>0),
        price int check (price>0)
    );

create table nav_movie.cinema_hall_seat
    (
        id int primary key,
        cinema_hall_id int check(cinema_hall_id>0) references nav_movie.cinema_hall(id),
        seat_id int check (seat_id>0) references nav_movie.seat(id)

    );


create table nav_movie.orders
    (
        id int primary key,
        price int,
        seat_id int check (seat_id>0) references nav_movie.seat(id),
        user_id int check (user_id>0) references nav_movie.users(id),
        session_id int check (session_id>0) references nav_movie.sessions(id)
    );
```

Триггеры

В процессе заполнения базы данных стало точно видно, какие атрибуты будут нуждаться в дополнительной проверке. Для этого был создан ряд триггеров, которые проверяли основные данные. Например, вот триггер для проверки фильмов при добавлении:

```
CREATE FUNCTION movie_trig() RETURNS trigger AS $movie_trig$
    BEGIN
        -- Не может быть такого, чтобы у фильма не было названия
        IF NEW.name IS NULL THEN
            RAISE EXCEPTION 'Movie should have a name';
        END IF;

        -- За компанию проверяем и айди
        IF NEW.id < 0 THEN
            RAISE EXCEPTION 'ID should be more than zero';
        END IF;

        -- Не может быть такого, чтобы у фильма не было названия
        IF NEW.duration IS NULL THEN
            RAISE EXCEPTION 'Duration can not be null';
        END IF;

        RETURN NEW;

    END;
$movie_trig$ LANGUAGE plpgsql;


CREATE TRIGGER movie_trig BEFORE INSERT OR UPDATE ON nav_movie.movie
    FOR EACH ROW EXECUTE PROCEDURE movie_trig();
```

Пример работы на тестовых данных: попытка загрузить отрицательный ID.

```
db_test.public> INSERT INTO nav_movie.test(id, test_name, test_date, test_time)
                    VALUES (-9, 'Alla', '2021-04-26', '6:23')
[2021-12-20 15:43:24] [P0001] ERROR: RABOTAY PLEASE cannot have a negative id
```

Описание механизмов обеспечения целостности данных

Для заполнения таблиц стоят ограничения в DDL коде на атрибуты. Например: ID не должен быть ниже нуля (нулевое ID допускается), цены не могут быть отрицательными, фильм обязан иметь название и ненулевую продолжительность. У каждой записи в таблицах есть уникальный идентификатор. В таблице клиентов есть атрибуты ID, имя и фамилия, которые подгружаются из Telegram и являются уникальными для всех пользователей.

Для покупки билета клиент должен пройти регистрацию. Без регистрации нет возможности просмотра сеансов и получения номера заказа. Данный механизм позволит избежать покупки билета пользователями, которых нет в базе данных, а также приобретения одного и того же билета двумя пользователями.

Разработка клиентского приложения

С помощью чат-бота клиентом может стать любой пользователь, который заинтересован в просмотре фильма в кинотеатре. Клиенту доступны следующие функции: выбор кинотеатра, поиск фильма, уведомления о выходе новых фильмов, выбор сеанса и зала, покупка билета, просмотр топа лучших фильмов за последнее время.

Базовые сценарии пользователя и команды

1. Регистрация – пользователь вводит команду **/reg** и бот автоматически сохраняет его ID, имя и фамилию из Telegram.
2. Поиск кинотеатра – пользователь вводит команду **/cinema**, на выход бот выводит список кинотеатров. Есть таковых нет – пишет соответствующее сообщение и делает шаг на предыдущее действие.
3. Поиск фильма – аналогично с поиском кинотеатра, по команде **/movie** пользователь получает список доступных фильмов. Есть таковых нет – пишет соответствующее сообщение и делает шаг на предыдущее действие.
4. Выбор сеанса – после выбора доступного фильма или кинотеатра, бот автоматически без дополнительных команд выводит список доступных сеансов. Есть таковых нет – пишет соответствующее сообщение и делает шаг на предыдущее действие.
5. Выбор места – аналогично с выбором сеанса, бот автоматически без дополнительных команд выводит список доступных мест в зале. Есть таковых нет – пишет соответствующее сообщение и делает шаг на предыдущее действие.
6. Покупка билета – после выбора доступного места пользователь получает сообщение со сгенерированным индивидуальным кодом билета, по которому сможет распечатать билет прямо в кинотеатре. Также приходит сообщение с точной информацией о заказе: выбранный кинотеатр, фильм, сеанс и сумма к оплате.
7. Просмотр топа фильмов – вне зависимости от покупки билета, любой пользователь с помощью команды **/popula**r может увидеть топ три фильма, основываясь на последних заказах других пользвователей.

Примеры ввода и редактирования данных в БД

Вся база данных заполнялась полностью вручную. Примеры запросов заполнения:

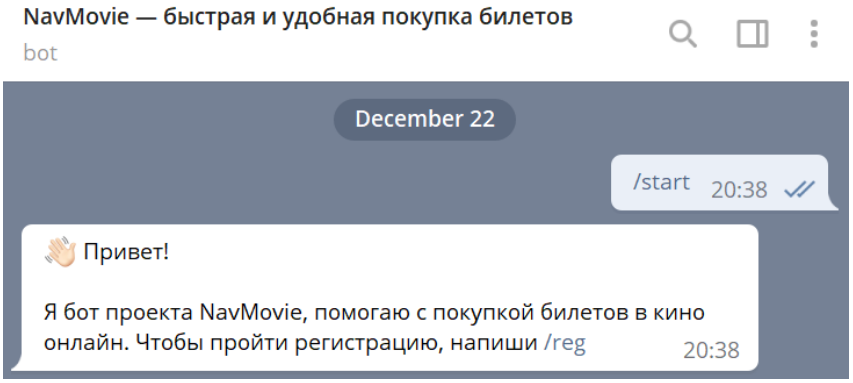
```
INSERT INTO nav_movie.movie(id, name, genre, duration, rental_start, rental_end)
    VALUES (1, 'Человек-паук: Нет пути домой', 'фантастика', '02:28', '2021-12-15', '2021-12-31');

INSERT INTO nav_movie.cinema(id, name, address) VALUES(1, 'Северный', 'ул. Пушкина д.7');

INSERT INTO nav_movie.hall(id, capacity, name) VALUES (2, 50, 'Красный');

INSERT INTO nav_movie.cinema_hall(id, cinema_id, hall_id) VALUES(3, 2, 3);

INSERT INTO nav_movie.sessions(id, cinema_hall_id, movie_id, session_date, session_time)
    VALUES(3, 3, 2, '2021-04-22', '14:00');

INSERT INTO nav_movie.seat(id, seat_row, seat_column, price) VALUES(7, 2, 9, 300)
```

Кроме того, при разработке бота на Python также были использованы SQL запросы, вот примеры работы с некоторыми функциями (полный код доступен на GitHub проекта):

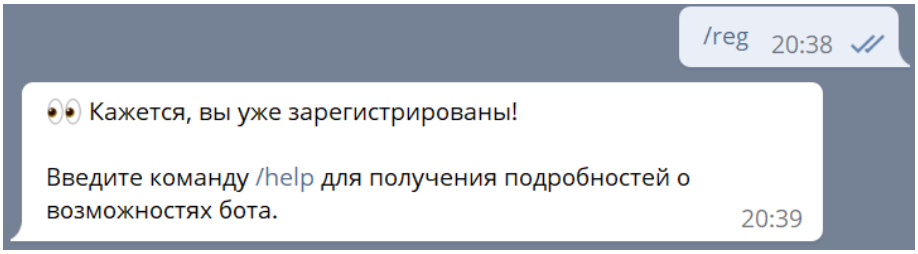
```
async def sessions(message: types.Message, state: FSMContext):
    user_data = await state.get_data()
    db = await connect()
    sessions_names = await db.fetch('''SELECT session_date, session_time, id FROM nav_movie.sessions WHERE
    nav_movie.sessions.movie_id = (SELECT id FROM nav_movie.movie WHERE nav_movie.movie.name = $1) AND
    nav_movie.sessions.cinema_hall_id IN ( SELECT id FROM nav_movie.cinema_hall WHERE nav_movie.cinema_hall.cinema_id =
    (SELECT id FROM nav_movie.cinema WHERE nav_movie.cinema.name = $2))''',
                                    user_data['choosen_movie'], user_data['choosen_cinema'])
    await db.close()
```

Тестирование готового бота

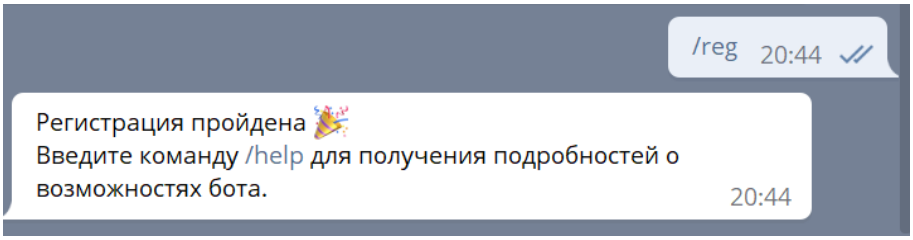
Первым шагом идет регистрация пользователя в системе. Для старта бота используется стандартный вызов **/start**:



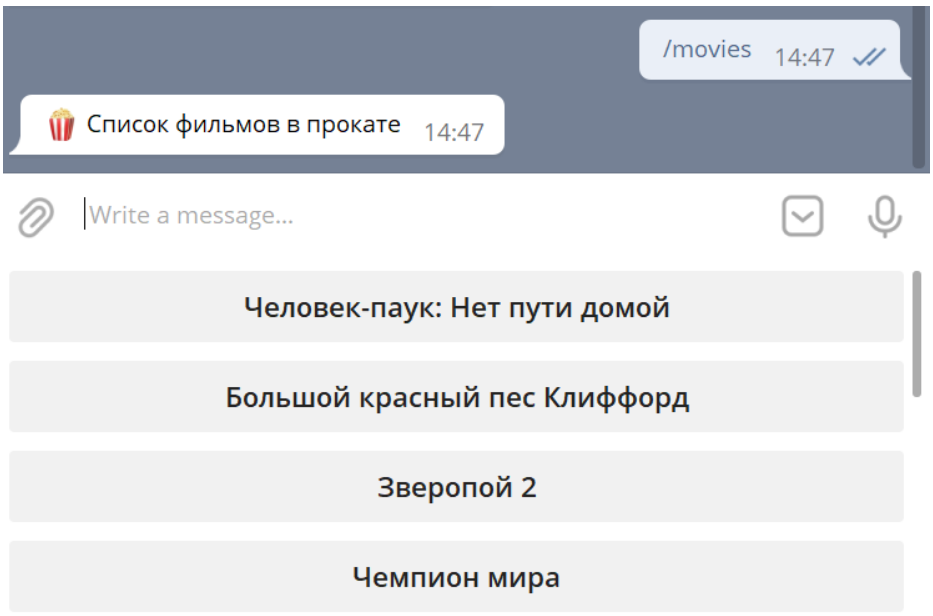
По предложенной команде **/reg** бот предлагает зарегистрироваться. Если зарегистрированный пользователей выберет эту команду, получит следующее сообщение:



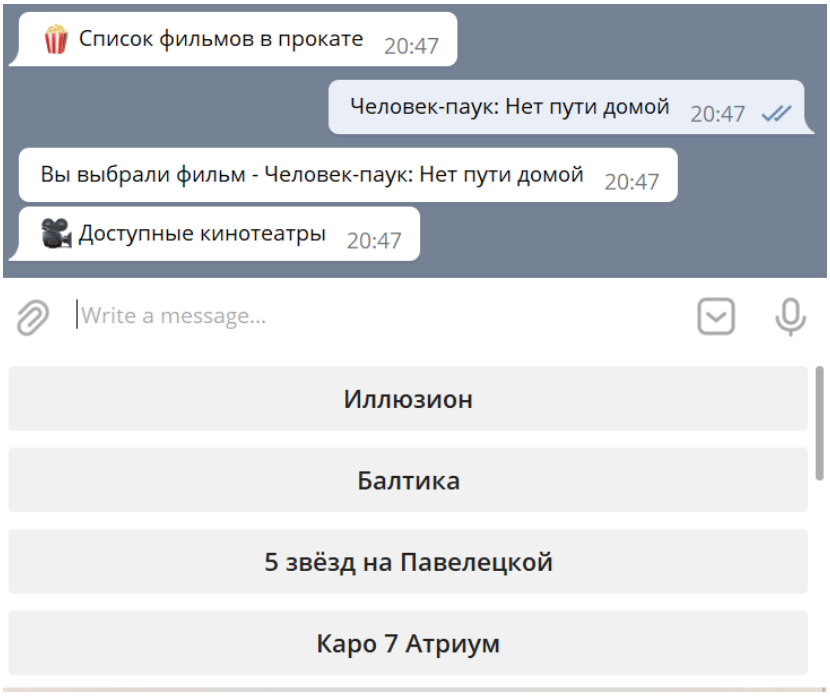
Если пользователя еще нет в базе данных, то бот внесет его ID, имя и фамилию самостоятельно. После чего предложит посмотреть функции бота:



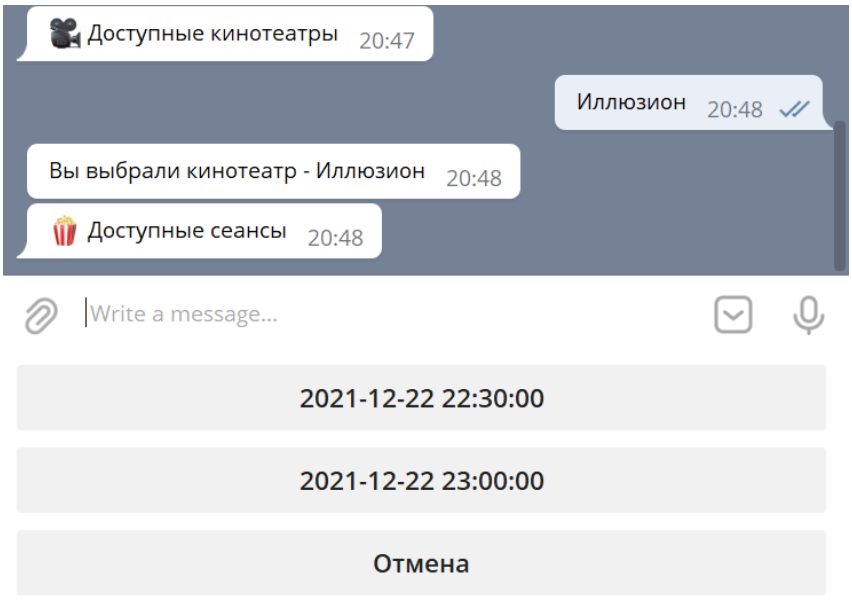
Для примера попробуем подобрать фильм. Бот выведет список всех доступных фильмов, которые в прокате и у которых сегодня есть сеансы, ниже с возможностью скроллинга. Выберем «Человек-паук: Нет пути домой».



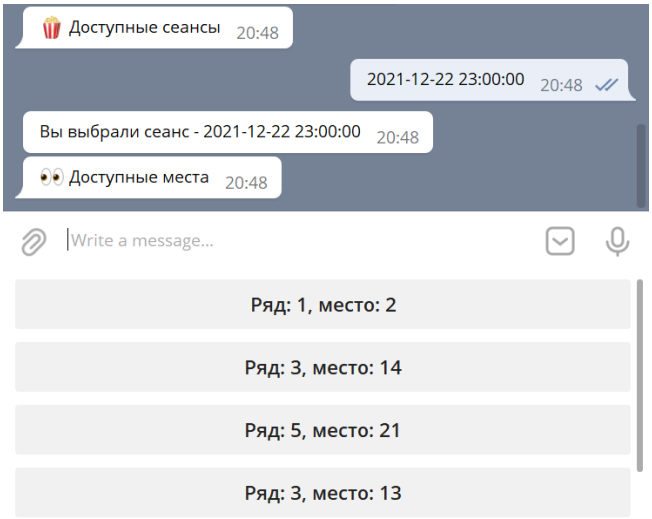
Предположим, что мы выбрали фильм. Далее бот предложит нам подобрать кинотеатр:



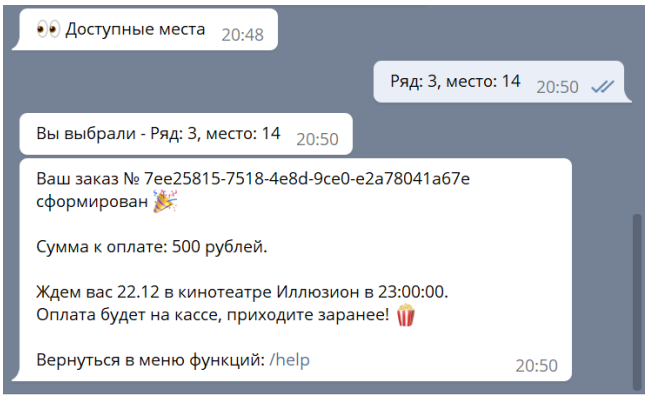
Пусть нам интересен кинотеатр «Иллюзион». Теперь бот нам предлагает доступные сеансы:



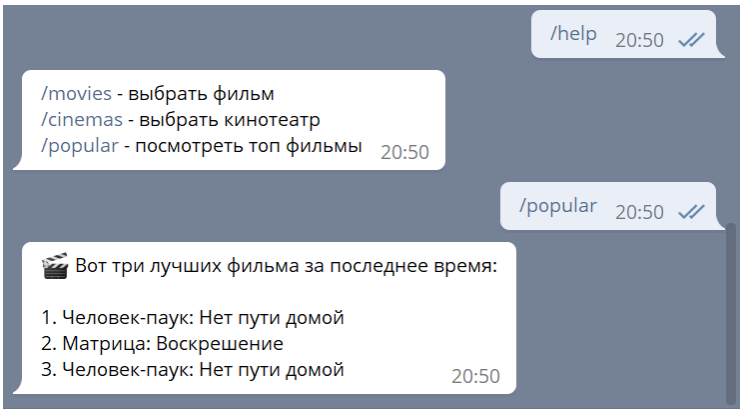
Выберем любой доступный сеанс. Далее переходим к выбору мест:



Отлично! После всей процедуры выбора бот сформировал уникальный номер и забронировал билет.



Можно купить билет на другой фильм, а можно посмотреть самые популярные по команде /popular:



Заключение

Проект реализовывался в течение нескольких месяцев. Большую часть времени велась разработка непосредственно бота, потому что это потребовало изучения дополнительной литературы, и бот создавался командой начиная с нуля знаний.

В процессе создания были закреплены на практике многие вещи из курса «Теория баз данных»: неоднократно были переделаны концептуальная и логическая модели, разобрано много вариантов версий DDL кода, проделана многочасовая работа по развертыванию базы данных в DataGtip. На наш взгляд, мы получили отличный опыт полноценной разработки и попробовали себя в роли реальных разработчиков.

Кроме того, возможна разработка дополнительной функциональности проекта, например:

1. Другие варианты анализа данных: в какое время чаще всего покупают билеты в кино, в какие дни недели и даты в месяце.
2. Возможность купить билет прямо внутри бота, с переадресацией пользователя на сайт банка.
3. Более точная информация о содержащихся данных: наличие свободных мест на сеансах, предлагать самые близкие к пользователю кинотеатры, сохранение билета внутри бота.
4. Выбор фильма по жанру, описанию, актеру.
5. И многое другое!

Ссылки

1. Гитхаб проекта: https://github.com/ansmirnova-mmtr/tbd_project
2. Код бота: https://github.com/ansmirnova-mmtr/tbd_project/blob/main/main.py

3. Видеопрезентация работы бота: https://github.com/ansmirnova-mmtr/tbd_project/blob/main/README.md

4. Концептуальная модель: https://lucid.app/lucidchart/d17b4a11-8a2e-4fae-bd04-f34adb4977bc/edit?invitationId=inv_a1a3e3ac-039b-42d4-8cc9-6149faa074f4
5. Логическая модель: https://lucid.app/lucidchart/43eee853-4de2-42f2-9fbc-f9684fac9314/edit?invitationId=inv_92e9535e-b0a5-4259-a86c-2232557ac97f