МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Базы данных»
Тема: Тестирование БД на безопасность.

Студент гр. 3384 Копасова К. А.
Преподаватель Михайлова С. А.

Санкт-Петербург
2025

Цель работы: провести тестирование базы данных на безопасность.

Задание:

Пусть требуется создать программную систему, предназначенную для работников справочной службы кинотеатров города. Такая система должна обеспечивать хранение сведений о кинотеатрах города, о фильмах, которые в них демонстрируются, о сеансах и билетах на эти сеансы. Сведения о кинотеатре — это его название, район города, где расположен кинотеатр, категория, вместимость. Сведения о фильме — это название фильма, режиссёр, оператор, актёры, сыгравшие главные роли, жанр; производство, наличие призов кинофестивалей, продолжительность сеанса, кадр из фильма для рекламы. Кроме того, должна храниться информация о репертуаре кинотеатров на месяц, то есть о том какие фильмы, когда и где демонстрируются, о ценах на билеты и о количестве свободных мест на тот или иной сеанс. На разных сеансах в одном кинотеатре могут идти разные фильмы, а если в кинотеатре несколько залов, то и на одном. Кинотеатр может ввести новый фильм в репертуар или убрать фильм из репертуара. Работник справочной службы может корректировать перечень фильмов, находящихся в прокате – добавлять новые фильмы и снимать с проката, а также перечень кинотеатров, поскольку кинотеатры могут открываться или закрываться, причём иногда временно, например, на ремонт. Цена билета определяется прокатной стоимостью копии фильма, сеансом и категорией кинотеатра. Справочной службе могут потребоваться следующие сведения о текущем состоянии проката фильмов в городе:

1. Репертуар кинотеатра?

2. Адрес и район кинотеатра ?

3. Число свободных мест на данный сеанс в указанном кинотеатре?

4. Цена билетов на данный сеанс в указанном кинотеатре?

5. Жанр, производство и режиссёр данного фильма ?

6. Какие фильмы имеют награды, когда и в каких кинотеатрах они демонстрируются?

7. В каких кинотеатрах в указанный день на указанных сеансах демонстрируется комедия?

Задачи:

- Сделать простой web-сервер для выполнения запросов из ЛР3, например с express.js. Не обязательно делать авторизацию и т.п., хватит одного эдпоинта на каждый запрос, с параметрами запроса как query parameters.
- Намеренно сделайте несколько (2-3) запроса, подверженных SQL-инъекциям
- Проверьте Ваше API с помощью sqlmap (или чего-то аналогичного), передав эндпоинты в качестве целей атаки. Посмотрите, какие уязвимости он нашёл (и не нашёл), опишите пути к исправлению.
- +2 балла, если напишете эндпоинт с уязвимостью, которая не находится sqlmap-ом.

Выполнение работы

Для выполнения лабораторной работы был реализован простой web-сервис на Python с использованием Flask и SQLAlchemy для работы с базой данных PostgreSQL.

Сервер представляет набор API-эндпоинтов для выполнения запросов, аналогичных тем, что использовались в лабораторной работе №3. Каждый эндпоинт принимает параметры через query parameters и возвращает данные в формате JSON с поддержкой русских символов. Стартовая страница содержит ссылки на запросы из работы №3 и формы для всех запросов, которые разделены на безопасные (5 запросов: 4 - 7) и на уязвимые к SQL-инъекциям (3 запроса: 1 - 3). Уязвимые эндпоинты используют прямую подстановку параметров в SQL-запрос, а безопасные - параметризированные запросы SQLAlchemy. Сервер также форматирует даты и время для удобного отображения и поддерживает передачу списков идентификаторов сеансов (запрос 7). Работает на порту 3000 в режиме отладки.

Проверим работу API с безопасным (см. рис. 1) и уязвимым (см. рис. 2) запросами.

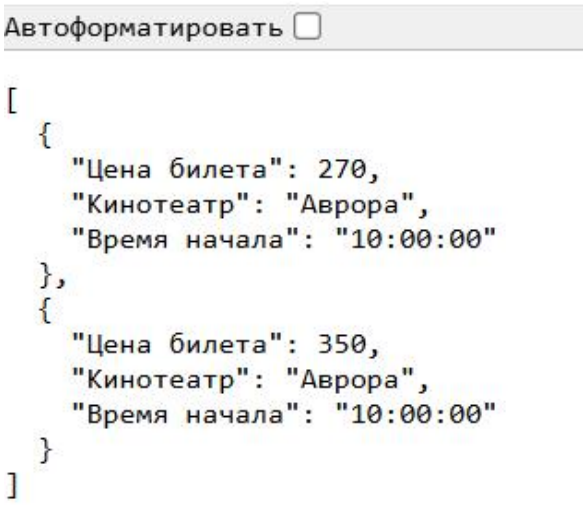

Рисунок 1 - Демонстрация безопасного запроса №4.

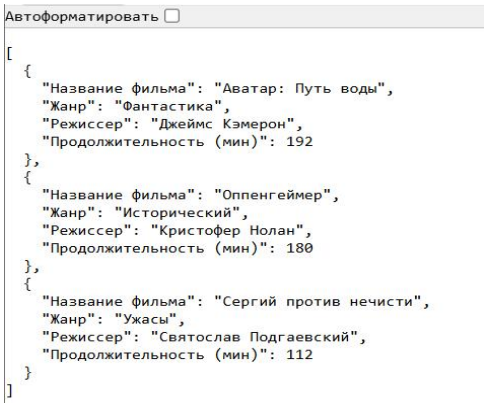

Рисунок 2 - Демонстрация уязвимого запроса №1.

Проверим API с помощью sqlmap, передав эндпоинты в качестве целей атаки.

1. Репертуар кинотеатра:

```
sqlmap -u "http://localhost:3000/repertoire?cinema_name=Аврора" --batch
```

Получаем следующий лог (рис. 3):

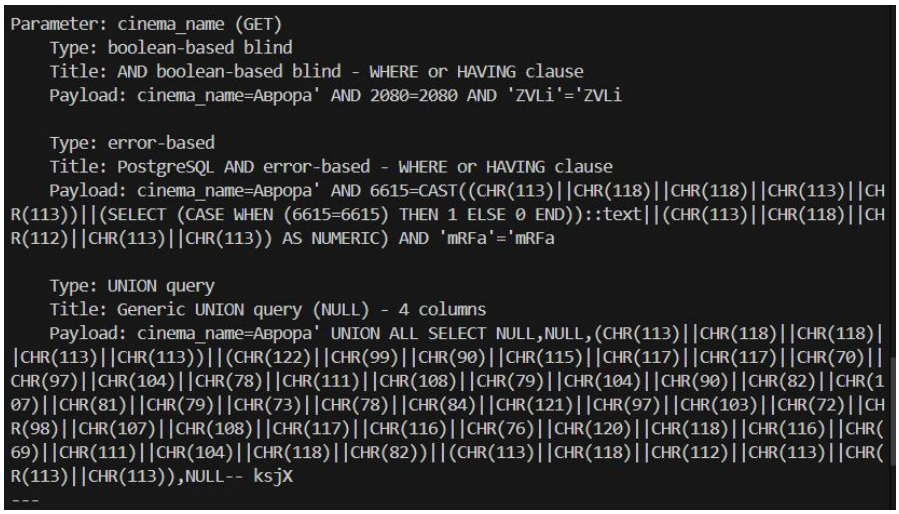

Рисунок 3 - SQL-инъекция обнаружена (параметр cinema_name).

Лог показывает, что sqlmap смог выполнить несколько типов вредных запросов через параметр cinema_name. Boolean-based означает, что сервер по-разному отвечает на запросы в зависимости от того, истинно или ложно подставленное условие. Error-based использует ошибки базы данных, через которые можно получить дополнительные данные. UNION-based позволяет добавлять в ответ сервера свои результаты, объединяя их с исходным запросом.

Появление всех этих типов в логе означает, что параметр действительно уязвим к SQL-инъекциям.

2. Адрес кинотеатра:

```
sqlmap -u "http://localhost:3000/cinema_address?cinema_name=Аврора" --batch
```

Получаем следующий лог (рис. 4):

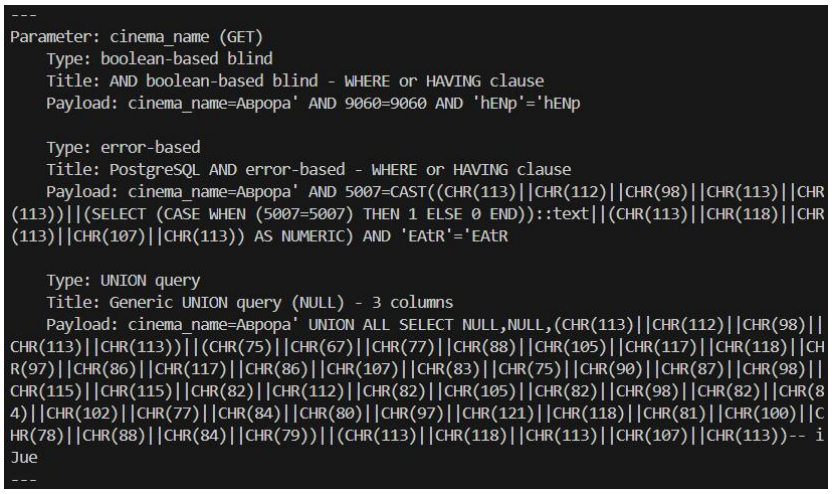

Рисунок 4 - SQL-инъекция обнаружена (параметр cinema_name).

Аналогичный лог получаем и при втором тестовом запросе, что также подтверждает наличие SQL-инъекции и уязвимость параметра.

3. Свободные места на сеанс:

```
sqlmap -u "http://localhost:3000/free_seats?session_id=1" --batch
```

Получаем следующий лог (рис. 5):

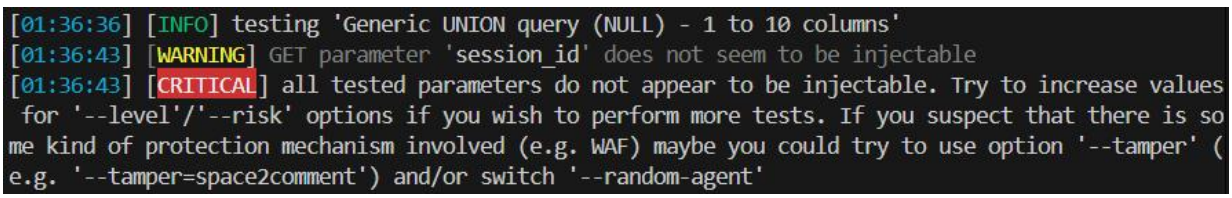

Рисунок 5 - SQL-инъекция не обнаружена (параметр session_id).

Тест sqlmap на параметре session_id не выявил SQL-инъекции. Несмотря на потенциально небезопасный код, параметр не поддаётся автоматическому внедрению вредоносных SQL-подстрок из-за ограничения типа данных.

4. Цены билетов на сеанс:

```
sqlmap -u
"http://localhost:3000/ticket_prices?cinema_name=Аврора&session_id=1" --batch
```

Получаем следующий лог (рис. 6, рис. 7):

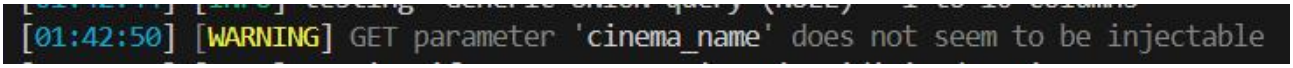

Рисунок 6 - SQL-инъекция не обнаружена (параметр cinema_name).

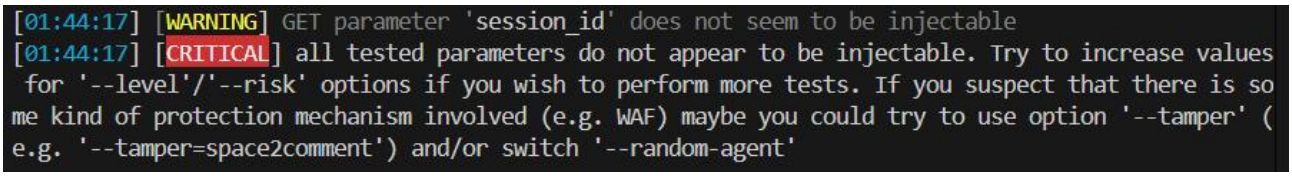

Рисунок 7 - SQL-инъекция не обнаружена (параметр session_id).

При проверке параметров cinema_name и session_id sqlmap последовательно протестировал оба параметра на наличие различных типов SQL-инъекций и ни один из параметров не оказался уязвим.

Запрос реализован через ORM-фреймворк SQLAlchemy, который автоматически параметризует значения и предотвращает выполнение произвольных SQL-команд, поэтому данный запрос безопасный.

5. Информация о фильме:

```
sqlmap -u "http://localhost:3000/film_info?film_title=Барби" --batch
```

Получаем следующий лог (рис. 8):

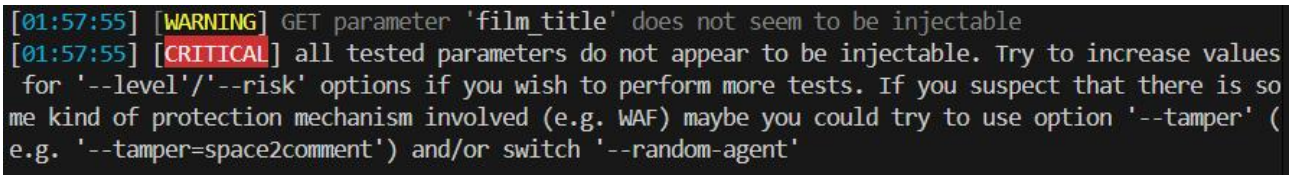

Рисунок 8 - SQL-инъекция не обнаружена (параметр film_title).

При проверке параметра film_title sqlmap также не обнаружил SQL-инъекций.

6. Фильмы с наградами:

В данном запросе отсутствуют пользовательские параметры, поэтому он не содержит точек ввода данных и не может быть уязвим для SQL-инъекций.

7. Комедии по дням и сеансам:

```
sqlmap -u "http://localhost:3000/comedy_on_day?date=2025-10-21&session_ids=1,2,3,6,7" --batch
```

Получаем следующий лог (рис. 9, рис. 10):

Рисунок 9 - SQL-инъекция не обнаружена (параметр date).

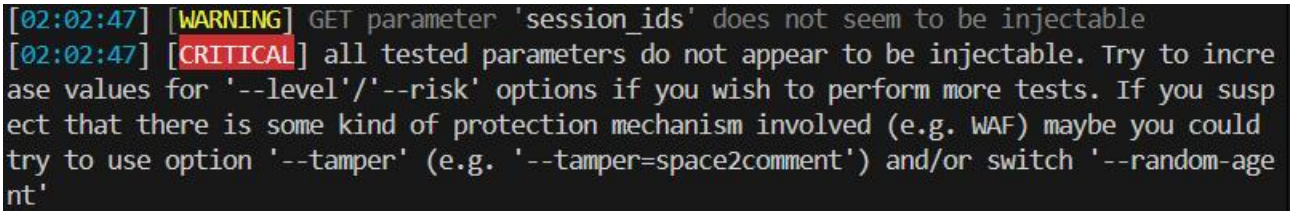

Рисунок 10 - SQL-инъекция не обнаружена (параметр session_ids).

При проверке параметров date и session_ids sqlmap не обнаружил SQL-инъекций.

В ходе проверки было выявлено, что уязвимости присутствуют только в эндпоинтах, где параметр cinema_name подставляется в SQL-запрос без защиты. Основная причина - формирование SQL вручную через конкатенацию строк.

Чтобы устранить уязвимости, нужно:

1) Перейти на параметризованные запросы или использовать ORM. При параметризации значения подставляются безопасно, и sqlmap не сможет внедрить свои конструкции.

2) Ограничивать формат входных данных.

Вывод

В ходе лабораторной работы проведено нагрузочное тестирование безопасности API с помощью инструмента sqlmap для оценки уязвимости эндпоинтов к SQL-инъекциям.

Проверка показала, что уязвимости присутствуют только в эндпоинтах, где параметр cinema_name подставляется в SQL-запрос напрямую через конкатенацию строк. Остальные параметры и запросы, включая session_id, film_title, date и session_ids, оказались защищенными, благодаря использованию параметризованных запросов и ORM (SQLAlchemy).

Для устранения уязвимостей рекомендуется использовать параметризованные запросы или ORM, ограничивать формат входных данных и права доступа к базе данных. Эти меры обеспечат безопасное взаимодействие с базой и защитят от SQL-инъекций.

ПРИЛОЖЕНИЕ 1

Ссылка на pull request: https://github.com/moevm/sql-2025-3384/pull/46.

Файл *server.py*:

```
from flask import Flask, request, jsonify
from flask.json.provider import DefaultJSONProvider
from sqlalchemy import create_engine, text
from datetime import datetime
import json

class RussJSON(DefaultJSONProvider):
    def dumps(self, obj, **kwargs):
        return json.dumps(obj, ensure_ascii=False, **kwargs)

    def loads(self, s, **kwargs):
        return json.loads(s, **kwargs)



DATABASE_URL = "postgresql+psycopg2://postgres:000000@localhost:5432/postgres"
engine = create_engine(DATABASE_URL)
app = Flask(__name__)
app.json = RussJSON(app)

# Стартовая страница
@app.route("/")
def index():
    return """
    <h1>API LAB5 — тестовая панель</h1>

    <h2>Эндпоинты напрямую (Запросы из ЛБ3)</h2>
    <ul>
        <li><a
href="/repertoire?cinema_name=Аврора">/repertoire?cinema_name=Аврора</a></li>
        <li><a
href="/cinema_address?cinema_name=Аврора">/cinema_address?cinema_name=Аврора</a>
</li>
        <li><a
href="/free_seats?session_id=1">/free_seats?session_id=1</a></li>
```

```
        <li><a
href="/ticket_prices?cinema_name=Аврора&session_id=1">/ticket_prices?cinema_name
=Аврора&session_id=1</a></li>

        <li><a
href="/film_info?film_title=Барби">/film_info?film_title=Барби</a></li>

        <li><a href="/films_with_prizes">/films_with_prizes</a></li>

        <li><a                                href="/comedy_on_day?date=2025-10-
21&session_ids=1,2,3,6,7">/comedy_on_day?date=2025-10-
21&session_ids=1,2,3,6,7</a></li>


    </ul>


    <h2>Уязвимые запросы</h2>


    <h3>1. репертуар кинотеатра</h3>
    <form action="/repertoire" method="get">
        <input name="cinema_name" placeholder="Название кинотеатра">
        <button type="submit">Отправить</button>
    </form>


    <h3>2. адрес кинотеатра</h3>
    <form action="/cinema_address" method="get">
        <input name="cinema_name" placeholder="Название кинотеатра">
        <button type="submit">Отправить</button>
    </form>


    <h3>3. свободные места на сеанс</h3>
    <form action="/free_seats" method="get">
        <input name="session_id" placeholder="ID сеанса" type="number">
        <button type="submit">Отправить</button>
    </form>


    <hr>


    <h2>Безопасные запросы</h2>


    <h3>4. цены билетов на сеанс</h3>
    <form action="/ticket_prices" method="get">
        <input name="cinema_name" placeholder="Название кинотеатра">
        <input name="session_id" placeholder="ID сеанса" type="number">
```

```
        <button type="submit">Отправить</button>
    </form>

    <h3>5. информация о фильме</h3>
    <form action="/film_info" method="get">
        <input name="film_title" placeholder="Название фильма">
        <button type="submit">Отправить</button>
    </form>

    <h3>6. фильмы с наградами</h3>
    <form action="/films_with_prizes" method="get">
        <button type="submit">Показать</button>
    </form>

    <h3>7. комедии по дням и сеансам</h3>
    <form action="/comedy_on_day" method="get">
        <input name="date" placeholder="Дата (YYYY-MM-DD)">
        <input name="session_ids" placeholder="ID сеансов (через запятую)">
        <button type="submit">Отправить</button>
    </form>

    <hr>

    """

# 1. репертуар кинотеатра (SQL-инъекция)
@app.route("/repertoire")
def repertoire():
    cinema_name = request.args.get("cinema_name", "")
    query = f"""
        SELECT DISTINCT f.title AS "Название фильма",
                        f.genre AS "Жанр",
                        f.director AS "Режиссер",
                        f.session_duration AS "Продолжительность (мин)"
        FROM films f
        JOIN sessions s ON f.film_id = s.film_id
        JOIN cinema_halls ch ON s.hall_id = ch.hall_id
        JOIN cinema c ON ch.cinema_id = c.cinema_id
```

```
        WHERE c.title = '{cinema_name}'        -- SQL-инъекция
    """
    with engine.connect() as conn:
        rows = [dict(r._mapping) for r in conn.execute(text(query))]
    return jsonify(rows)

# 2. адрес кинотеатра (SQL-инъекция)
@app.route("/cinema_address")
def cinema_address():
    cinema_name = request.args.get("cinema_name", "")
    query = text(f"""
        SELECT title AS "Кинотеатр",
               city_region AS "Район",
               address_cinema AS "Адрес"
        FROM cinema
        WHERE title = '{cinema_name}'
    """)
    with engine.connect() as conn:
        rows = [dict(r._mapping) for r in conn.execute(query)]
    return jsonify(rows)

# 3. свободные места на сеанс (SQL-инъекция)
@app.route("/free_seats")
def free_seats():
    session_id = request.args.get("session_id", type=int)

    query = text(f"""
        SELECT COUNT(*) AS "Свободные места"
        FROM tickets t
        JOIN sessions s USING(session_id)
        WHERE s.session_id = {session_id} AND t.sold_out = '0';
    """)
    with engine.connect() as conn:
        result = conn.execute(query, {"sid": session_id})
        row = result.fetchone()

    return jsonify(row._asdict())
```

```
# 4. цены билетов на сеанс (безопасный запрос)
@app.route("/ticket_prices")
def get_ticket_prices():
    cinema_name = request.args.get("cinema_name", "")
    session_id = request.args.get("session_id", type=int)
    query = """
        SELECT DISTINCT
            t.cost AS "Цена билета",
            c.title AS "Кинотеатр",
            TO_CHAR(s.start_time, 'HH24:MI:SS') AS "Время начала"
        FROM tickets t
        JOIN sessions s ON t.session_id = s.session_id
        JOIN cinema_halls ch ON s.hall_id = ch.hall_id
        JOIN cinema c ON ch.cinema_id = c.cinema_id
        WHERE c.title = :cinema_name AND s.session_id = :session_id
    """
    with engine.connect() as conn:
        result = conn.execute(text(query), {"cinema_name": cinema_name,
"session_id": session_id})
        rows = [dict(r._mapping) for r in result]
    return jsonify(rows)


# 5. информация о фильме (безопасный запрос)
@app.route("/film_info")
def film_info():
    film_title = request.args.get("film_title", "")
    query = text("""
        SELECT title AS "Фильм",
               genre AS "Жанр",
               production AS "Производство",
               director AS "Режиссер"
        FROM films
        WHERE title = :film_title
    """)

    with engine.connect() as conn:
        row = conn.execute(query, {"film_title": film_title}).fetchone()
```

```
        return jsonify(dict(row._mapping) if row else {})

    # 6. фильмы с наградами (безопасный запрос)
    @app.route("/films_with_prizes")
    def get_films_with_prizes():
        query = """
            SELECT
                f.title AS "Фильм",
                p.title AS "Награда",
                fest.title AS "Фестиваль",
                TO_CHAR(s.date_session, 'YYYY-MM-DD') AS "Дата сеанса",
                TO_CHAR(s.start_time, 'HH24:MI:SS') AS "Время начала",
                c.title AS "Кинотеатр",
                ch.title AS "Зал",
                c.address_cinema AS "Адрес"
            FROM films f
            JOIN prizes p ON f.film_id = p.film_id
            JOIN festival fest ON p.festival_id = fest.festival_id
            JOIN sessions s ON f.film_id = s.film_id
            JOIN cinema_halls ch ON s.hall_id = ch.hall_id
            JOIN cinema c ON ch.cinema_id = c.cinema_id
        """
        with engine.connect() as conn:
            result = conn.execute(text(query))
            rows = [dict(r._mapping) for r in result]
        return jsonify(rows)

    # 7. комедии по дням и сеансам (безопасный запрос)
    @app.route("/comedy_on_day")
    def comedy_on_day():
        date_str = request.args.get("date", "")
        session_ids_str = request.args.get("session_ids", "")

        date_obj = datetime.strptime(date_str, "%Y-%m-%d").date() if date_str else
None

        session_ids = []
```

```
    if session_ids_str:
        session_ids = [int(sid.strip()) for sid in session_ids_str.split(",") if sid.strip().isdigit()]

    if not date_obj or not session_ids:
        return jsonify({"error": "Необходимо указать дату и ID сеансов"}), 400

    query = text("""
        SELECT c.title "Кинотеатр",
               f.title "Фильм",
               f.genre "Жанр",
               s.session_id "Сеанс",
               ch.title "Название зала",
               TO_CHAR(s.date_session, 'YYYY-MM-DD') AS "Дата сеанса",
               TO_CHAR(s.start_time, 'HH24:MI:SS') AS "Время начала",
               TO_CHAR(s.end_time, 'HH24:MI:SS') AS "Окончание сеанса"
        FROM sessions s
        JOIN films f USING(film_id)
        JOIN cinema_halls ch USING(hall_id)
        JOIN cinema c USING(cinema_id)
        WHERE f.genre = 'Комедия'
            AND s.date_session = :date
            AND s.session_id = ANY(:session_ids);
    """)

    with engine.connect() as conn:
        result = conn.execute(
            query,
            {"date": date_obj, "session_ids": session_ids}
        )
        rows = [dict(row._mapping) for row in result]
    return jsonify(rows)

if __name__ == "__main__":
    app.run(debug=True, port=3000)
```