

## Лабораторная работа

МДК.06.04 Интеллектуальные системы и технологии

### Ассистент кинотеатра «Луч» на Next.js с собственным MCP-сервером

#### Цель работы

Пройти весь путь создания веб-приложения с чат-ассистентом: MCP-сервер на Python хранит данные и даёт инструменты, приложение на Next.js показывает данные и чат, языковая модель в Ollama вызывает инструменты сервера. Экзаменационное задание устроено так же, отличается только предметная область. Поэтому в лабораторной важны порядок работы и приёмы, а не афиша кинотеатра.

Продолжительность — 4 академических часа (две пары).

Объяснений здесь минимум. У каждого шага указаны разделы курсов «MCP-серверы» и «Next.js», где тема разобрана подробно. Запись «MCP 7.5» означает курс «MCP-серверы», раздел 7.5, запись «Next.js 14.5» — курс «Next.js», раздел 14.5.

#### Что получится

| Часть                   | Где работает                                                                 | Что делает                                                                                                                                                          |
|-------------------------|------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MCP-сервер</b>       | Python,<br><a href="http://127.0.0.1:8000/mcp">http://127.0.0.1:8000/mcp</a> | хранит афишу и брони в памяти; инструменты <code>list_sessions</code> , <code>book_tickets</code> , <code>cancel_booking</code> и ресурс <code>afisha://week</code> |
| <b>Страница «Афиша»</b> | Next.js, <a href="http://localhost:3000">http://localhost:3000</a>           | читает ресурс без участия модели и показывает сеансы со свободными местами                                                                                          |

| Часть                | Где работает     | Что делает                                                           |
|----------------------|------------------|----------------------------------------------------------------------|
| Страница «Ассистент» | Next.js, /chat   | чат: администратор пишет обычным языком, модель вызывает инструменты |
| Модель               | Ollama, qwen3:4b | читает описания инструментов и решает, какой вызвать                 |

Браузер не обращается к MCP-серверу напрямую: с ним работает только серверная часть Next.js.

## Подготовка

- Python 3.14 и файл requirements.txt из курса «MCP-серверы» (MCP 1.5).
- Node.js 24 (Next.js 1.2).
- Ollama со скачанной моделью qwen3:4b (MCP 4.2).
- MCP Inspector, установленный по разделу MCP 1.7.

Проект состоит из двух папок рядом: mcp-server и web.

## Ход работы

### Шаг 1. Окружение сервера

Создайте папку mcp-server, скопируйте в неё requirements.txt из курса и поставьте пакеты. На Windows в PowerShell:

```
cd mcp-server
py -m venv .venv
.venv\Scripts\Activate.ps1
pip install -r requirements.txt
```

На macOS и Linux:

```
cd mcp-server
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

**Подробнее:** MCP 1.5 «Готовим окружение».

## Шаг 2. Данные сервера

Файл `mcp-server/server.py`, начало — афиша, брони и функции поиска сеанса:

```
"""MCP-сервер кинотеатра «Луч»: афиша на неделю и бронь билетов."""
import json
import logging
import sys
from typing import Annotated

from pydantic import Field
from mcp.server import MCPServer
from mcp.server.mcpserver.exceptions import ToolError

# Журнал — в поток ошибок, а не в print (раздел 3.3 курса)
logging.basicConfig(stream=sys.stderr, level=logging.INFO)
mcp = MCPServer("Кинотеатр «Луч»")

DAYS = ["понедельник", "вторник", "среда", "четверг",
        "пятница", "суббота", "воскресенье"]

# Афиша на неделю. Брони живут в памяти процесса:
# после перезапуска сервера они пропадают.
SESSIONS = [
    {"day": "пятница", "time": "19:00", "film": "Дюна",
     "hall": 1, "seats": 4},
    {"day": "пятница", "time": "21:30", "film": "Интерстеллар",
     "hall": 2, "seats": 6},
    {"day": "суббота", "time": "12:00", "film": "Чебурашка",
     "hall": 1, "seats": 8},
    {"day": "суббота", "time": "18:00", "film": "Дюна",
     "hall": 2, "seats": 2},
    {"day": "воскресенье", "time": "15:00", "film": "Чебурашка",
     "hall": 2, "seats": 5},
    {"day": "воскресенье", "time": "20:00", "film": "Интерстеллар",
     "hall": 1, "seats": 3},
]

# Брони: номер сеанса в списке -> {имя клиента: число билетов}
BOOKINGS: dict[int, dict[str, int]] = {
    i: {} for i in range(len(SESSIONS))
}

def check_day(day: str) -> str:
    """Приводит день к виду из DAYS или отказывает с подсказкой."""
    day = day.strip().lower()
    if day not in DAYS:
        raise ToolError(f"Неизвестный день «{day}». "
                        f"Допустимо: {' '.join(DAYS)}.")
    return day

def find_session(day: str, film: str) -> int:
    """Номер сеанса по дню и названию фильма."""
    day = check_day(day)
```

```

title = film.strip(" «»\").lower()
for i, s in enumerate(SESSIONS):
    if s["day"] == day and s["film"].lower() == title:
        return i
films = [s["film"] for s in SESSIONS if s["day"] == day]
raise ToolError(f"Фильма «{film}» нет в афише на день «{day}». "
                f"В этот день: {' '.join(films) or 'ничего'}.")

def free_seats(i: int) -> int:
    return SESSIONS[i]["seats"] - sum(BOOKINGS[i].values())

def with_free(i: int) -> dict:
    return {**SESSIONS[i], "free": free_seats(i)}

```

- Брони хранятся в памяти процесса: база данных не нужна, но после перезапуска сервера брони пропадают.
- Журнал пишется в поток ошибок: print в MCP-сервере опасен.
- check\_day и find\_session отказывают через ToolError. Текст отказа получит модель, поэтому он объясняет, что исправить, и перечисляет допустимые значения.
- Название фильма сравнивается без учёта регистра и кавычек: модель может прислать и «Дюна», и дюна.

**Подробнее:** MCP 1.6 «Первый сервер», 3.3 «Куда выводить отладку: почему print опасен», 7.5 «Текст ошибки как интерфейс для модели», 7.7 «Защитное программирование внутри инструмента».

### Шаг 3. Инструменты и ресурс

Продолжение того же файла:

```

@mcp.tool()
def list_sessions(day: str) -> list[dict]:
    """Сеансы на день недели со свободными местами.

    day — день недели по-русски: «пятница», «суббота».
    """
    day = check_day(day)
    return [with_free(i) for i, s in enumerate(SESSIONS)
            if s["day"] == day]

@mcp.tool()
def book_tickets(
    day: str,
    film: str,
    client_name: str,
    count: Annotated[int, Field(ge=1, le=6)] = 1,
) -> str:

```

```

"""Бронирует билеты на сеанс.

day — день недели, film — название фильма из афиши,
client_name — имя клиента в именительном падеже,
count — число билетов, от 1 до 6.
"""

i = find_session(day, film)
name = client_name.strip()
if not name:
    raise ToolError("Укажите имя клиента.")
if name in BOOKINGS[i]:
    raise ToolError(f"У клиента {name} уже есть бронь "
                    "на этот сеанс. Сначала отмените её.")
free = free_seats(i)
if free == 0:
    raise ToolError("На этот сеанс свободных мест нет.")
if count > free:
    raise ToolError(f"Свободно мест: {free}, запрошено: {count}.")
BOOKINGS[i][name] = count
s = SESSIONS[i]
return (f"Бронь: {name}, «{s['film']}», {s['day']} {s['time']}, "
        f"зал {s['hall']}, билетов: {count}.")

@mcp.tool()
def cancel_booking(day: str, film: str, client_name: str) -> str:
    """Отменяет бронь клиента на сеанс.

    day — день недели, film — название фильма,
    client_name — имя клиента в именительном падеже.
    """

    i = find_session(day, film)
    name = client_name.strip()
    if name not in BOOKINGS[i]:
        raise ToolError(f"У клиента {name} нет брони на этот сеанс.")
    count = BOOKINGS[i].pop(name)
    return f"Бронь {name} отменена, освобождено мест: {count}."

@mcp.resource("afisha://week")
def afisha() -> str:
    """Афиша на неделю со свободными местами, JSON."""
    rows = [with_free(i) for i in range(len(SESSIONS))]
    return json.dumps(rows, ensure_ascii=False)

if __name__ == "__main__":
    mcp.run(transport="streamable-http")

```

- Модель видит имя инструмента, docstring и типы аргументов. Поэтому в docstring сказано, в каком виде передавать день и имя клиента.
- Инструменты брони принимают день и название фильма, а не номер сеанса. С номером модели пришлось бы сначала искать его через list\_sessions, и маленькие модели на этом ошибаются.

- Ограничение Field(ge=1, le=6) попадает в схему инструмента: модель заранее видит, что билетов бывает от одного до шести.
- Ресурс afisha://week читает само приложение, чтобы показать афишу, а инструменты вызывает модель.
- transport="streamable-http" запускает сервер в сетевом режиме на адресе 127.0.0.1 и порту 8000.

Запустите сервер командой `python server.py`.

**Подробнее:** MCP 2.5 «Docstring как контракт», 2.6 «Что инструмент возвращает», 7.3 «Ограничения прямо в сигнатуре», 5.1 «Кто нажимает кнопку: инструмент, ресурс или промпт», 5.2 «Статический ресурс и выбор URI-схемы», 3.5 «Streamable HTTP: поднимаем сервер по сети».

#### Шаг 4. Проверка сервера без модели

Во втором терминале запустите MCP Inspector командой `mcp-inspector` и откройте адрес, который он напечатает. Добавьте сервер вручную (в версии 2.10.1 — Add Servers, затем Add manually): транспорт Streamable HTTP, адрес `http://127.0.0.1:8000/mcp`. Подключитесь к нему и проверьте:

- `list_sessions` с днём «суббота» возвращает два сеанса;
- `list_sessions` с днём «в субботу» отвечает ошибкой со списком допустимых дней;
- `book_tickets` на «Дюну» в субботу на два билета проходит, а бронь другого клиента на тот же сеанс получает отказ «свободных мест нет»;
- `cancel_booking` для клиента без брони отвечает отказом;
- ресурс `afisha://week` возвращает JSON с полем `free`.

Инструмент, который не работает в Inspector, с моделью тоже не заработает. Консольный режим Inspector для быстрой проверки описан в разделе MCP 3.7.

**Подробнее:** MCP 1.7 «Отладка без ИИ: MCP Inspector», 3.7 «Подключаемся к сетевому серверу клиентом».

#### Шаг 5. Модель

Если модель ещё не скачана, скачайте её и проверьте, что она умеет вызывать инструменты:

```
ollama pull qwen3:4b
ollama show qwen3:4b
```

В разделе Capabilities вывода `ollama show` должна быть строка `tools`.

**Подробнее:** MCP 4.2 «Ставим Ollama и проверяем, что она работает», 4.5 «Выбор модели».

## Шаг 6. Проект Next.js и подключение к серверу

Рядом с папкой `mcp-server` создайте проект и поставьте пакеты AI SDK (команды одинаковы на Windows, macOS и Linux):

```
npx create-next-app@16.3.0 web --yes
cd web
npm install --save-exact ai@7.0.133 @ai-sdk/react@4.0.136
npm install --save-exact @ai-sdk/mcp@2.0.71 ai-sdk-ollama@4.4.0
```

Содержимое `app/globals.css` замените одной строкой:

```
@import "tailwindcss";
```

Стили шаблона лежат вне слоёв Tailwind и перекрывают утилиты, а в тёмной теме системы перекрашивают текст страницы.

Файл `web/lib/mcp.ts` — подключение к серверу:

```
import "server-only";
import { createMCPClient } from "@ai-sdk/mcp";

// Один клиент на один запрос: открыть, поработать, закрыть
export async function openMcp() {
  const url = process.env.MCP_URL ?? "http://127.0.0.1:8000/mcp";
  return createMCPClient({ transport: { type: "http", url } });
}
```

Строка `import "server-only"` остановит сборку, если файл по ошибке импортирует клиентский компонент: MCP-клиент работает только на сервере Next.js.

**Подробнее:** Next.js 1.2 «Установка и создание проекта», 1.4 «Структура проекта», 5.8 «DTO — Data Transfer Objects» (заметка о пакете `server-only`), 14.5 «AI Streaming с Vercel AI SDK» (установка).

## Шаг 7. Шапка и навигация

Файл `web/app/layout.tsx` — общий каркас всех страниц:

```
import type { Metadata } from "next";
import Link from "next/link";
import NavLinks from "@components/NavLinks";
```

```

import "../globals.css";

export const metadata: Metadata = {
  title: "Кинотеатр «Луч»",
  description: "Афиша и ассистент администратора кинотеатра",
};

export default function RootLayout({
  children,
}: LayoutProps<"/">) {
  return (
    <html lang="ru">
      <body className="min-h-screen bg-slate-50 text-slate-900">
        {/* Шапка: название слева, меню справа */}
        <header className="border-b border-slate-200 bg-white">
          <nav className="mx-auto flex max-w-4xl items-center justify-between gap-4 p-4">
            <Link href="/" className="text-lg font-bold">
              Кинотеатр «Луч»
            </Link>
            <NavLinks />
          </nav>
        </header>
        {children}
      </body>
    </html>
  );
}

```

Файл web/components/NavLinks.tsx — меню с подсветкой текущей страницы:

```

"use client";

import Link from "next/link";
import { usePathname } from "next/navigation";

const LINKS = [
  { href: "/", label: "Афиша" },
  { href: "/chat", label: "Ассистент" },
];

// Клиентский компонент: текущий адрес знает только браузер
export default function NavLinks() {
  const pathname = usePathname();
  return (
    <ul className="flex gap-4">
      {LINKS.map((link) => (
        <li key={link.href}>
          <Link
            href={link.href}
            className={
              pathname === link.href
                ? "font-semibold text-indigo-700"
                : "text-slate-600 hover:text-slate-900"
            }
          >
            {link.label}
          </Link>
        </li>
      ))}
    </ul>
  );
}

```

```

        </Link>
      </li>
    )})
  </ul>
);
}

```

Текущий адрес знает только браузер, поэтому меню вынесено в отдельный клиентский компонент, а layout остаётся серверным.

**Подробнее:** Next.js 1.8 «Вложенные маршруты и layout.tsx», 4.1 «Компонент Link и навигация», 4.3 «Практика: навигация с активными ссылками и поиском».

## Шаг 8. Страница «Афиша»

Файл web/app/page.tsx:

```

import { openMcp } from "@lib/mcp";

// Страница строится на каждый запрос: брони меняются в чате
export const dynamic = "force-dynamic";

type Session = {
  day: string;
  time: string;
  film: string;
  hall: number;
  seats: number;
  free: number;
};

// Читает ресурс afisha://week без участия модели
async function loadAfisha(): Promise<Session[]> {
  const mcp = await openMcp();
  try {
    const res = await mcp.readResource({ uri: "afisha://week" });
    const first = res.contents[0];
    return typeof first?.text === "string"
      ? JSON.parse(first.text)
      : [];
  } finally {
    await mcp.close();
  }
}

export default async function AfishaPage() {
  const sessions = await loadAfisha();
  const days = [...new Set(sessions.map((s) => s.day))];
  return (
    <main className="mx-auto max-w-4xl p-4">
      <h1 className="mb-4 text-2xl font-bold">Афиша на неделю</h1>
      {days.map((day) => (
        <section key={day} className="mb-6">
          <h2 className="mb-2 text-lg font-semibold capitalize">

```

```

    {day}
  </h2>
  <ul className="grid gap-3 sm:grid-cols-2">
    {sessions
      .filter((s) => s.day === day)
      .map((s) => (
        <li
          key={s.time + s.film}
          className={`rounded-lg border p-4 ${
            s.free === 0
              ? "border-red-300 bg-red-50"
              : "border-slate-200 bg-white"
            }`}
        >
          <p className="font-semibold">
            {s.time} – {s.film}
          </p>
          <p className="text-sm text-slate-600">
            Зал {s.hall}
          </p>
          <p className="text-sm">
            {s.free === 0
              ? "Мест нет"
              : `Свободно ${s.free} из ${s.seats}`}
          </p>
        </li>
      )
    )
  </ul>
</section>
))}
</main>
);
}

```

- Страница серверная: ресурс читается на сервере Next.js методом `readResource`, модель в этом не участвует.
- `dynamic = "force-dynamic"` заставляет строить страницу на каждый запрос. Без этой строки Next.js соберёт её один раз при сборке, и брони из чата на ней не появятся.
- Клиент открывается на один запрос и закрывается в `finally`, даже если чтение упало.
- Сеанс без мест выделен красной рамкой и фоном.

**Подробнее:** Next.js 2.1 «Серверные vs клиентские компоненты», 5.1 «Fetch в серверных компонентах», 11.1 «Стратегии рендеринга»; МСР 5.3 «Ресурс со стороны клиента»; Tailwind CSS 5.3 «Адаптивные сетки и автосетки».

## Шаг 9. Страница ошибки

Если МСР-сервер не запущен, `openMcp` бросает исключение. Его перехватывает файл `web/app/error.tsx`:

```
"use client";
```

```
// Сюда попадает исключение со страницы, например
// когда MCP-сервер не запущен
export default function Error({ retry }: { retry: () => void }) {
  return (
    <main className="mx-auto max-w-4xl p-4">
      <h1 className="text-xl font-bold text-red-700">
        Афиша недоступна
      </h1>
      <p className="my-2">
        MCP-сервер не отвечает. Запустите его и нажмите «Повторить».
      </p>
      { /* retry заново запрашивает страницу с сервера */ }
      <button
        onClick={() => retry()}
        className="rounded bg-indigo-600 px-4 py-2 text-white"
      >
        Повторить
      </button>
    </main>
  );
}
```

Кнопка вызывает `retry`: он заново запрашивает страницу у сервера, и после запуска MCP-сервера афиша появляется без перезагрузки. В статьях вместо `retry` встречается `reset`, но он перерисовывает страницу из уже полученных данных и снова показывает ошибку.

**Подробнее:** Next.js 1.11 «`loading.tsx`, `error.tsx`, `not-found.tsx`».

## Шаг 10. Обработчик чата

Файл `web/app/api/chat/route.ts`:

```
import {
  convertToModelMessages, stepCountIs, streamText, type UIMessage,
} from "ai";
import { ollama } from "ai-sdk-ollama";
import { openMcp } from "@lib/mcp";

const SYSTEM = [
  "Ты администратор кинотеатра «Луч». Отвечай только по-русски.",
  "Для броней и вопросов об афише используй инструменты.",
  "День недели передавай в именительном падеже: «суббота».",
  "Имя клиента передавай в именительном падеже: «Петров».",
].join(" ");

export async function POST(req: Request) {
  const { messages }: { messages: UIMessage[] } = await req.json();
  const mcp = await openMcp().catch(() => null);
  if (!mcp) {
    // этот текст useChat покажет в поле error
    return new Response("MCP-сервер не запущен", { status: 503 });
  }
  const result = streamText({
    model: ollama(process.env.LLM_MODEL ?? "qwen3:4b"),
```

```

    system: SYSTEM,
    messages: await convertToModelMessages(messages),
    tools: await mcp.tools(),
    stopWhen: stepCountIs(5),
    onFinish: () => mcp.close(),
    onError: () => mcp.close(),
  });
  return result.toUIMessageStreamResponse();
}

```

- Обработчик открывает MCP-клиент и передаёт модели все инструменты сервера (mcp.tools()).
- Системный промпт задаёт роль, требует русский язык и говорит, в каком виде передавать день и имя.
- stepCountIs(5) ограничивает число шагов модели: каждый вызов инструмента — отдельный шаг, и ответ после него — ещё один.
- Клиент закрывается и при завершении ответа, и при ошибке.
- Если сервер не запущен, обработчик отвечает кодом 503 с текстом, который страница чата покажет как ошибку.
- Модель меняется без правки кода: строка LLM\_MODEL=qwen3:1.7b в файле web/.env.local.

**Подробнее:** Next.js 8.1 «Что такое Route Handlers», 14.5 «AI Streaming с Vercel AI SDK» (серверная часть), 13.6 «Переменные окружения»; MCP 4.4 «Мост: соединяем MCP-сервер и Ollama», 4.6 «Системный промпт: обязательная строка про русский язык».

## Шаг 11. Страница «Ассистент»

Файл web/app/chat/page.tsx:

```

"use client";

import { useChat } from "@ai-sdk/react";
import { useState } from "react";

// Так выглядит результат MCP-инструмента
type ToolOutput = {
  content?: { text?: string }[];
  isError?: boolean;
};

export default function ChatPage() {
  const [input, setInput] = useState("");
  const { messages, sendMessage, status, error } = useChat();
  const busy = status === "submitted" || status === "streaming";

  return (
    <main className="mx-auto max-w-2xl p-4">
      <h1 className="mb-4 text-2xl font-bold">Ассистент</h1>

```



```
    />
    <button
      className="rounded bg-indigo-600 px-4 text-white
                disabled:opacity-50"
      disabled={busy}
    >
      Отправить
    </button>
  </form>
</main>
);
}
```

Ответ модели приходит частями сообщения:

| Тип части    | Что это                                        | Что делает страница                 |
|--------------|------------------------------------------------|-------------------------------------|
| text         | текст ответа                                   | показывает абзацем                  |
| dynamic-tool | вызов инструмента: toolName, state и результат | показывает строкой «имя: результат» |
| reasoning    | рассуждение модели                             | не показывает                       |

Отказ ToolError приходит не состоянием output-error, а обычным результатом (output-available), у которого isError равно true. Поэтому страница проверяет оба случая и красит отказ красным. Пока status равен submitted или streaming, поле ввода и кнопка заблокированы.

**Подробнее:** Next.js 14.5 «AI Streaming с Vercel AI SDK» (клиентская часть), 2.3 «Директива "use client" — правила и нюансы».

Шаг 12. Сценарий проверки и линтер

Запустите три процесса: Ollama, MCP-сервер (python server.py в папке mcp-server) и приложение (npm run dev в папке web). Пройдите сценарий:

- 1. Страница «Афиша» показывает три дня по два сеанса.
- 2. В чате: «Что идёт в субботу?» — модель вызывает list\_sessions и перечисляет два сеанса.
- 3. «Забронируй два билета на Дюну в субботу на имя Петров» — вызов book\_tickets, бронь оформлена.
- 4. Обновите «Афишу»: у субботней «Дюны» мест нет, карточка выделена.
- 5. «Забронируй билет на Дюну в субботу для Ивановой» — отказ «свободных мест нет» виден в ленте красным, модель сообщает его.

6. «Отмени бронь Петрова на Дюну в субботу» — вызов `cancel_booking`, места освободились.
7. Остановите MCP-сервер: «Афиша» показывает страницу ошибки, чат — «MCP-сервер не запущен». Запустите сервер и нажмите «Повторить».
8. Сузьте окно до 380px: шапка, карточки и чат помещаются без горизонтальной прокрутки.

Если модель не вызывает инструмент, пройдите чек-лист из раздела MCP 4.9. В конце проверьте код линтером:

```
npm run lint
```

**Подробнее:** MCP 4.9 «Модель не вызывает инструмент: чек-лист диагностики»; Tailwind CSS 6.2 «Брейкпоинты» (как проверять адаптивность в браузере).

### Что показал проверочный прогон

- На компьютере, где готовилась работа, ответ с вызовом инструмента на `qwen3:4b` занимал от 40 секунд до минуты. На слабом компьютере он может идти дольше.
- Модель каждый раз вызывала нужный инструмент с верными аргументами. Получив отказ «свободных мест нет», она сама заглянула в афишу и посоветовала выбрать другой сеанс.
- «Отмени бронь Петрова» дошло до сервера как «Петров» — помогла строка промпта про именительный падеж. Но у той же строки есть обратная сторона: «на имя Соколова» модель превратила в «Соколов». На экзамене эту задачу нужно решить самостоятельно: промптом, мягким сравнением имён на сервере или бронью по телефону.

### Проверка результата

Пройдите по списку. Он повторяет критерии экзаменационного задания.

- Сервер: афиша, ресурс и три инструмента; каждый неверный ввод получает отказ `ToolError` с понятным текстом.
- «Афиша» читает ресурс через MCP-клиент, сеансы без мест выделены, вёрстка держится на 380px.
- Модель вызывает инструменты, каждый вызов виден в ленте чата, пока модель думает, ввод заблокирован.

- При остановленном сервере обе страницы показывают понятное сообщение.
- В TypeScript нет any, у инструментов есть аннотации типов и docstring, npm run lint проходит без ошибок.

### **Контрольные вопросы**

1. Почему браузер не обращается к МСР-серверу напрямую?
2. Чем ресурс отличается от инструмента и почему афишу страница читает ресурсом?
3. Почему инструменты брони принимают день и название фильма, а не номер сеанса?
4. Что увидит модель, если инструмент бросит ToolError, и как это показывает страница чата?
5. Зачем странице «Афиша» строка dynamic = "force-dynamic"?
6. Что ограничивает stopWhen: stepCountIs(5) и что будет без этого ограничения?
7. Почему рассуждение модели не выводится на экран?
8. Что делает retry в error.tsx и чем он отличается от reset?