MCP szerverek és LLM integráció

Ez a rész azt mutatja be, hogyan lehet egy nagy nyelvi modellt szabályozott módon összekapcsolni külső rendszerekkel MCP szerverek segítségével. A fókusz a modern integrációs szemléleten van: nem az a cél, hogy az LLM közvetlenül érje el a legacy rendszereket, hanem az, hogy egy szabályozott, egységes interfészen keresztül kapjon hozzáférést adatokhoz és műveletekhez.

Az MCP rövidítés jelentése: **Model Context Protocol**.

Miért érdekes ez rendszerintegrációban?

Sok szervezetnél már léteznek olyan rendszerek, amelyek üzletileg kritikusak, de technológiailag régebbiek:

- vállalatirányítási rendszerek;
- belső SQL adatbázisok;
- fájlszerverek;
- dokumentumtárak;
- régebbi SOAP vagy REST szolgáltatások;
- egyedi fejlesztésű, legacy alkalmazások.

Ha egy LLM-et szeretnénk ezekkel együtt használni, akkor gyorsan felmerülnek a következő kérdések:

- honnan szerezzen adatot a modell;
- milyen műveleteket hajthat végre;
- milyen jogosultságokkal férhet hozzá a rendszerekhez;
- hogyan lehet korlátozni és naplózni a műveleteit;
- hogyan lehet egységesíteni a sokféle háttérrendszer elérését.

Erre ad egy modern, egységes megközelítést az MCP.

Mi az MCP lényege?

Az MCP egy szabványos protokoll arra, hogy egy kliensalkalmazás strukturált módon összekapcsolódjon olyan szerverekkel, amelyek adatot, promptokat vagy végrehajtható műveleteket tesznek elérhetővé egy LLM számára.

Egyszerűen megfogalmazva:

- az **LLM** nem közvetlenül beszél a háttérrendszerrel;
- a **kliensalkalmazás** kezeli a modellt és a felhasználói interakciót;
- az **MCP szerver** egy jól definiált felületen keresztül elérhetővé teszi a szükséges adatokat és funkciókat.

Az MCP fő szereplői

1. Host alkalmazás

Ez az a program, amelyben a felhasználó ténylegesen dolgozik. Például:

- egy AI asszisztens;
- egy fejlesztői környezet;
- egy belső vállalati chat alkalmazás;
- egy saját integrációs felület.

Ez a komponens jeleníti meg a választ a felhasználónak, és ez kapcsolja össze az LLM-et az MCP világával.

2. MCP kliens

Az MCP kliens a host alkalmazás része vagy annak egy komponense. Feladata:

- kapcsolatot létesíteni az MCP szerverrel;
- lekérdezni, milyen képességeket kínál a szerver;
- átadni a modellnek a releváns eszközöket és adatokat;
- továbbítani a szerver felé a műveletkéréseket.

3. MCP szerver

Az MCP szerver teszi elérhetővé a háttérrendszerek képességeit egységes formában. Nem maga a legacy rendszer az MCP szerver, hanem egy olyan adapter vagy köztes réteg, amely becsomagolja a háttérrendszer funkcióit.

Például:

- egy adatbázis-lekérdező szolgáltatás;
- egy dokumentumtár adapter;
- egy CRM vagy ERP rendszerhez kapcsolódó integrációs réteg;
- egy ticketing rendszer műveleteit kiajánló szerver.

Mit tud egy MCP szerver biztosítani?

Az MCP három különösen fontos szerveroldali fogalmat használ.

1. Tools

A tool egy végrehajtható művelet. Ilyen lehet például:

- ügyfél adatainak lekérdezése azonosító alapján;
- rendelési státusz lekérdezése;
- hibajegy létrehozása;
- jelentés generálása;
- belső API meghívása.

Lényeges, hogy a tool nem csak adatot ad vissza, hanem valamilyen műveletet is képviselhet.

2. Resources

A resource olyan adat vagy tartalom, amelyet a kliens és a modell kontextusként felhasználhat. Például:

- dokumentáció;
- konfiguráció;
- kézikönyv;
- adatbázisból lekért strukturált adatok;
- riportok vagy logfájlok tartalma.

Ez különösen hasznos akkor, ha az LLM-nek először információt kell olvasnia és értelmeznie, mielőtt választ adna.

3. Prompts

A prompt ebben a kontextusban nem egyszerűen egy szabad szöveg, hanem egy újrahasznosítható sablon vagy workflow. Például:

- “elemezd ezt a logfájlt”;
- “készíts összefoglalót a kiválasztott hibajegyről”;
- “magyarázd el a kiválasztott SQL lekérdezés működését”.

Ez segít abban, hogy a gyakori feladatok szabványos módon jelenjenek meg a kliensben.

Egy tipikus vállalati példa

Tegyük fel, hogy egy vállalatnál van:

- egy régi ügyviteli rendszer;
- egy SQL adatbázis;
- egy dokumentumtár;
- és egy belső AI asszisztens.

Egy felhasználó ezt kérdezi:

- “Mondd meg, miért csúszik a 48291-es rendelés teljesítése.”

Ilyenkor egy lehetséges folyamat:

- a host alkalmazás elküldi a kérdést az LLM-nek;
- a kliens látja, hogy elérhető például egy `get_order_status` tool;
- az LLM meghívja ezt a toolt;
- az MCP szerver a háttérben lekérdezi a legacy rendszert;
- az eredményt strukturált formában visszaadja;
- az LLM ebből természetes nyelvű választ fogalmaz.

Ha szükséges, a modell további toolokat is használhat, például:

- `get_customer_notes`
- `get_delivery_events`
- `search_manual`

Így az LLM nem “kitalálja” az adatokat, hanem valós rendszerekből dolgozik.

Miért hasznos az MCP legacy rendszerek esetén?

1. Nem kell az LLM-et közvetlenül a háttérrendszerre engedni

Az MCP szerver egy köztes réteg. Ez biztonságosabb és jobban ellenőrizhető.

2. Egységes interfészt ad

Az LLM vagy a kliens szempontjából mindegy lehet, hogy a háttérben SQL, REST, SOAP vagy fájlrendszer van. Az MCP szerver egységesen hirdeti a képességeit.

3. Könnyebb jogosultságot és naplózást kezelni

Az integrációs logikát nem a modellbe építjük, hanem a szerveroldali komponensbe.

4. Jobban karbantartható

Ha megváltozik a háttérrendszer API-ja, elég lehet az MCP szervert módosítani, és nem kell az egész kliensoldali megoldást újratervezni.

Milyen kommunikációs elemek vannak a háttérben?

Az MCP koncepcionálisan egy szabványos kliens-szerver kommunikáció. A specifikáció a képességek egységes leírását és a hívások szabályos szerkezetét adja meg.

Oktatási szempontból a legfontosabb gondolatok:

- a kliens először felderíti, mire képes a szerver;

- a szerver deklarálja, milyen képességei vannak;
- a kliens ezeket a képességeket átadhatja a modellnek;
- a modell vagy a felhasználó ezek alapján kezdeményez műveleteket.

Hogyan néz ki egy egyszerű integrációs gondolkodásmód?

Ha egy legacy rendszert szeretnénk LLM-mel összekapcsolni, akkor érdemes a következő lépésekben gondolkodni:

1. Azonosítsuk a hasznos üzleti műveleteket

Nem azt kell kitenni, hogy “itt a teljes adatbázis”, hanem azt, hogy:

- milyen kérdésekre akarunk választ kapni;
- milyen műveleteket akarunk engedélyezni;
- milyen információkat kell a modell számára elérhetővé tenni.

2. Ezeket bontsuk le toolokra és resource-okra

Példák:

- tool: rendelési státusz lekérdezés;
- tool: hibajegy létrehozás;
- resource: termékdokumentáció;
- resource: belső folyamatleírás.

3. Korlátozzuk a hozzáférést

Nem minden művelet való automatikus modellhasználatra. Egyes tooloknál emberi jóváhagyás kellhet.

4. Gondoljunk a hibatűrésre

Mi történik, ha:

- a legacy rendszer nem elérhető;
- hibás adatot ad vissza;
- a felhasználó pontatlanul fogalmaz;
- több rendszer válaszát kell egyesíteni.

Egyszerű fogalmi példa

Tegyük fel, hogy van egy egyetemi tanulmányi rendszer. Az MCP szerver az alábbi toolokat teheti

elérhetővé:

- `get_student_data(neptunKod)`
- `get_course_list(neptunKod)`
- `get_exam_results(neptunKod, felev)`
- `create_helpdesk_ticket(szoveg)`

Lehetséges resource-ok:

- hallgatói szabályzat;
- tantárgyleírások;
- vizsgaszabályok;
- ügyintézési útmutatók.

Lehetséges prompt:

- “Fogalmazz udvarias választ a hallgatónak a vizsgajelentkezési problémájára a szabályzat alapján.”

Ez már egy teljes, modern integrációs minta: az LLM nem csak beszélget, hanem szabályozott módon kapcsolódik valódi szervezeti tudáshoz és műveletekhez.

Kipróbálható Python példa

Az alábbi példa egy minimális, kipróbálható MCP szervert mutat be Python nyelven. A cél nem egy teljes vállalati megoldás, hanem annak szemléltetése, hogy egy legacy jellegű háttérrendszer képességeit hogyan lehet toolok, resource-ok és promptok formájában kiajánlani.

Ez a példa a jelenlegi, 2026. március 27-én ellenőrzött Python MCP SDK szemléletéhez igazodik.

Mit csinál a példa?

- van egy egyszerű MCP szerver;
- a szerver két toolt kínál:
  - hallgatói alapadatok lekérdezése;
  - vizsgaeredmények lekérdezése;
- van egy resource is, amely egy rövid vizsgaszabályzati szöveget ad vissza;
- van egy promptsablon is;
- a kliens csatlakozik a szerverhez és közvetlenül meghív két toolt.

Példafájlok

- `mcp_demo_server.py`
- `mcp_demo_client.py`

MCP szerver kód

```
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("Legacy Student MCP", json_response=True)

STUDENTS = {
    "ABC123": {
        "name": "Kiss Anna",
        "program": "Mernokinformatikus BSc",
        "status": "aktiv",
        "semester": 4,
    },
    "XYZ987": {
        "name": "Nagy Bela",
        "program": "Programtervezo informatikus BSc",
        "status": "aktiv",
        "semester": 6,
    },
}

EXAM_RESULTS = {
    ("ABC123", "2025-26-1"): [
        {"course": "Halozatok", "grade": 4},
        {"course": "Adatbazisok", "grade": 5},
    ],
    ("XYZ987", "2025-26-1"): [
        {"course": "Szoftvertechnologia", "grade": 3},
        {"course": "Mesterseges intelligencia", "grade": 5},
    ],
}

@mcp.tool()
def get_student_data(neptun_code: str) -> dict:
    """Hallgatoi alapadatok lekerdezese Neptun-kod alapjan."""
    student = STUDENTS.get(neptun_code.upper())
    if student is None:
        return {"error": "Nincs ilyen hallgato."}
    return {"neptun_code": neptun_code.upper(), **student}

@mcp.tool()
def get_exam_results(neptun_code: str, semester: str) -> dict:
    """Vizsgaeredmenyek lekerdezese egy adott felevre."""
    key = (neptun_code.upper(), semester)
    results = EXAM_RESULTS.get(key)
    if results is None:
        return {"error": "Nincs talalat a megadott hallgatohoz vagy
felevhez."}
```

```
    return {
        "neptun_code": neptun_code.upper(),
        "semester": semester,
        "results": results,
    }

@mcp.resource("guide://exam-rules")
def exam_rules() -> str:
    """Rovid vizsgaszabalyzati minta."""
    return (
        "A hallgato a vizsgara a hivatalos tanulmanyi rendszeren keresztul
jelentkezhet. "
        "Vizsgaidoszakban a jelentkezes es a lejelentkezes hataridokhoz
kotott. "
        "Problema eseten a hallgato helpdesk jegyet nyithat."
    )

@mcp.prompt()
def answer_student_question(question: str) -> str:
    """Promptsablon hallgatoi ugyintezesi kerdesek megvalaszolasahoz."""
    return (
        "Valaszolj roviden, targyszeruen es udvariasan a hallgatoi kerdesre.
"
        "Ha szukseges, hasznald a vizsgaszabalyzatot es a kapcsolodo
toolokat. "
        f"Kerdes: {question}"
    )

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

Egyszerű Python kliens

```
import asyncio
import sys
from pathlib import Path

from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client

async def main() -> None:
    server_script = Path(__file__).with_name("mcp_demo_server.py")
    server_params = StdioServerParameters(
        command=sys.executable,
        args=[str(server_script)],
    )
```

```
    async with stdio_client(server_params) as (read_stream, write_stream):
        async with ClientSession(read_stream, write_stream) as session:
            await session.initialize()

            tools_result = await session.list_tools()
            print("Elerheto toolok:")
            for tool in tools_result.tools:
                print(f"- {tool.name}: {tool.description}")

            student_result = await session.call_tool(
                "get_student_data",
                {"neptun_code": "ABC123"},
            )
            print("\\nget_student_data valasz:")
            print(student_result.content)

            exam_result = await session.call_tool(
                "get_exam_results",
                {"neptun_code": "ABC123", "semester": "2025-26-1"},
            )
            print("\\nget_exam_results valasz:")
            print(exam_result.content)

if __name__ == "__main__":
    asyncio.run(main())
```

Futtatás

Ha a `mcp` csomag még nincs telepítve:

```
python -m venv .venv
.venv\Scripts\activate
pip install mcp
```

Ezután a kliens futtatása:

```
python mcp_demo_client.py
```

Ha a kliens sikeresen kapcsolódik, akkor:

- kilistázza az elérhető toolokat;
- meghívja a `get_student_data` toolt;
- meghívja a `get_exam_results` toolt;
- kiírja a visszakapott strukturált választ.

Mit érdemes megfigyelni?

- A szerver nem egy teljes tanulmányi rendszer, hanem annak MCP adaptere.
- A kliens nem közvetlenül adatbázist hív, hanem a szerver által publikált toolokat.
- A háttérrendszer lecserélhető úgy, hogy a kliensoldali gondolkodás változatlan marad.
- A prompt és a resource külön fogalom a toolokhoz képest.

Mire kell figyelni tervezéskor?

1. Biztonság

- milyen adatot érhet el az LLM;
- milyen műveletet indíthat;
- kell-e emberi jóváhagyás;
- hogyan naplózzuk a hívásokat.

2. Adatminőség

Ha a háttérrendszer hibás vagy hiányos adatot ad, az LLM válasza is gyenge lesz.

3. Jogosultságkezelés

Nem biztos, hogy minden felhasználó minden toolhoz hozzáférhet.

4. Promptolási fegyelem

Érdemes világosan meghatározni:

- mit csináljon a modell;
- mikor használjon toolt;
- hogyan hivatkozzon a visszakapott adatokra;
- mikor mondja azt, hogy nincs elég információja.

Gyakorló kérdések

- Milyen legacy rendszert érdemes toolként becsomagolni, és mit érdemes inkább csak resource-ként elérhetővé tenni?
- Milyen műveleteknél szükséges emberi jóváhagyás?
- Miért nem jó ötlet a teljes belső adatbázist közvetlenül az LLM-nek adni?
- Hogyan lehet egy MCP szerverrel egységesíteni több különböző belső rendszer elérését?
- Milyen hibák jelenhetnek meg az integrációs láncban a felhasználói kérdéstől a háttérrendszerig?

Összefoglalás

Az MCP szerver egy modern integrációs minta LLM-ek számára.

Legfontosabb tanulságok:

- az MCP szerver köztes réteg a modell és a háttérrendszerek között;
- egységesen tesz elérhetővé műveleteket, adatokat és promptokat;
- különösen hasznos belső és legacy rendszerek LLM-alapú integrációjánál;
- a technikai integráció mellett a biztonság, jogosultság és adatminőség is központi kérdés.

További információ

Hivatalos MCP dokumentáció:

- Model Context Protocol specification
- MCP prompts
- MCP tools