Lab 8: Tworzenie Agentów AI MCP

Model Context Protocol

Materiały na kolokwium - Część 2

7 grudnia 2025

Spis treści

1 Model Context Protocol (MCP)

1.1 Podstawy

Czym jest MCP?

Model Context Protocol - otwarty protokół komunikacji między aplikacjami AI a źródłami danych/narzędziami
Problem który rozwiązuje:

- LLM mają ograniczony kontekst (training data cutoff)
- Brak dostępu do real-time data
- Brak dostępu do prywatnych danych (bazy danych, internal APIs)
- Brak możliwości wykonania akcji w external systems
- Każdy vendor ma własne API (vendor lock-in)

Rozwiązanie MCP:

- **Standardowy protokół** - jeden standard dla wszystkich
- **Modułowe rozszerzenia** - plug-and-play tools
- **Dostęp do external data** - real-time info
- **Możliwość akcji** - wywołanie funkcji, API calls
- **Provider-agnostic** - działa z różnymi LLM

Specyfikacja: modelcontextprotocol.io (open standard)

1.2 Architektura MCP

Komponenty Systemu

MCP Server:

- Udostępnia **tools** (narzędzia do wywołania)
- Udostępnia **resources** (dane, dokumenty)
- Udostępnia **prompts** (prompt templates)
- Implementacja: C#, Python, TypeScript, Go
- Hosting: lokalnie (stdio) lub zdalnie (HTTP)

MCP Client:

- Aplikacja AI (GitHub Copilot, ChatGPT, Claude, Gemini)
- Łączy się z MCP server
- Discover tools (lista dostępnych narzędzi)
- Invoke tools (wywołanie funkcji)
- Process responses

Transport Layer:

- **stdio** - standard input/output (local processes)
- **HTTP** - REST API (remote servers)
- **SSE** - Server-Sent Events (streaming)

Workflow MCP

Typowy przepływ:

1. User zadaje pytanie AI assistant
2. AI assistant analizuje - potrzebuje external data/action
3. Client wysyła request do MCP Server: "list tools"
4. Server zwraca listę dostępnych tools (name, description, parameters)
5. AI wybiera odpowiednie tool
6. Client wywołuje tool z parametrami
7. Server wykonuje akcję (query database, call API, read file)
8. Server zwraca wynik
9. AI przetwarza wynik i odpowiada user

Przykład:

- User: "Ile mamy klientów z Polski?"
- AI: potrzebuję danych z bazy
- Tool: `queryDatabase(sql: "SELECT COUNT(*) FROM customers WHERE country='Poland'")`
- Result: `{count: 1523}`
- AI: "Mamy 1523 klientów z Polski."

1.3 Porównanie z innymi podejściami

MCP vs Function Calling vs Plugins

Function Calling (OpenAI):

- LLM wywołuje funkcje zdefiniowane w aplikacji
- Vendor-specific API (tylko OpenAI)
- Tight coupling z providerem
- JSON schema dla function definitions
- Limited to OpenAI models

Plugins (ChatGPT):

- OpenAPI-based extensions
- Tylko ChatGPT
- Limited scope (HTTP only)
- Deprecated (zastępowane GPTs i Actions)

MCP (Model Context Protocol):

- **Universal standard** - open protocol
- **Provider-agnostic** - działa z OpenAI, Anthropic, Google, Microsoft
- **Modular** - plug-and-play servers
- **Multiple transports** - stdio, HTTP, SSE
- **Community-driven** - open source implementations
- **Richer capabilities** - tools + resources + prompts

Kiedy MCP:

- Multi-provider support (nie jesteś locked do jednego LLM)
- Reusable tools (share między różnymi AI apps)
- Complex integrations (databases, APIs, files)
- Local development (VS Code Copilot)

2 MCP Server

2.1 Podstawy Serwera

Rola MCP Server

Serwer udostępnia 3 typy zasobów:
1. Tools (Narzędzia):

- Funkcje które LLM może wywołać
- Przykłady: searchDatabase, sendEmail, readFile, callAPI
- Parametry wejściowe (input schema)
- Typ zwracany (output)

2. Resources (Zasoby):

- Dane dostępne dla LLM jako kontekst
- Przykłady: dokumentacja, config files, knowledge base
- Read-only (LLM czyta, nie modyfikuje)

3. Prompts (Szablony):

- Pre-defined prompt templates
- Parametryzowane prompty
- Przykład: "Analyze code in {file}"template

2.2 Implementacja w C# (.NET 9)

MCP SDK dla . NET

Instalacja:

- NuGet package: (sprawdź GitHub - dotnet-sdk)
- Repository: github.com/modelcontextprotocol/dotnet-sdk

Podstawowa struktura:

- `McpServer` - main server class
- `[McpTool]` - attribute dla tool methods
- `[McpResource]` - attribute dla resources
- Transport configuration (stdio/HTTP)

Tool definition:

- Public method z `[McpTool]` attribute
- Description (pomaga LLM decide kiedy użyć)
- Parameters z JSON Schema validation
- Return type (string, object, list)

Kontrakt Narzędzi

Tool metadata:

- **Name** - unikalna nazwa (camelCase recommended)
- **Description** - jasny opis co robi (LLM używa do decision)
- **Parameters** - input schema (JSON Schema)
- **Required fields** - które parametry obowiązkowe

JSON Schema dla parametrów:

- Type: string, number, boolean, object, array
- Format: date, email, uri, etc.
- Constraints: minLength, maxLength, pattern (regex)
- Enum: ograniczony zestaw wartości
- Validation przed wykonaniem tool

Przykład tool:

- Name: `searchDatabase`
- Description: "Search customer database by country"
- Parameters: `{country: string (required)}`
- Returns: `List<Customer>`

2.3 Transport: stdio vs HTTP

stdio Transport

Standard Input/Output
Charakterystyka:

- Local process communication
- Server runs jako subprocess
- Communication przez stdin/stdout
- JSON-RPC over stdio

Kiedy używać:

- Local development (VS Code extensions)
- Desktop applications
- Trusted environment (same machine)
- Low latency (no network overhead)

VS Code integration:

- Copilot automatycznie discovers MCP servers
- Configuration w `settings.json`
- Server path i arguments

Zalety:

- (+) Prosty setup
- (+) No authentication needed (local trust)
- (+) Low latency

Wady:

- (-) Only local (nie remote)
- (-) Wymaga spawning processes

HTTP Transport

REST API
Charakterystyka:

- Remote server communication
- Standard HTTP requests (POST)
- JSON payloads
- Can be hosted anywhere (Azure, AWS, on-prem)

Kiedy używać:

- Remote access (distributed systems)
- Multiple clients (shared server)
- Production deployments
- Cloud-hosted services

Security:

- HTTPS (TLS encryption)
- API key authentication
- Rate limiting
- CORS configuration

Zalety:

- (+) Remote access
- (+) Scalable (load balancing)
- (+) Language agnostic (any HTTP client)

Wady:

- (-) Network latency
- (-) Authentication complexity
- (-) Requires hosting infrastructure

3 MCP Client

3.1 Integracja z Aplikacją

Rola MCP Client

Client responsibilities:

- **Connect** - nawiązanie połączenia z MCP server
- **Discover** - listowanie dostępnych tools/resources
- **Invoke** - wywoływanie tools z parametrami
- **Handle responses** - przetwarzanie wyników
- **Error handling** - retry, timeout, fallback

Przykładowi klienci:

- GitHub Copilot (VS Code)
- Claude Desktop (Anthropic)
- Custom aplikacje (C#, Python, TypeScript)

Implementacja w C#

HttpClient approach:

- HttpClient do komunikacji z HTTP server
- JSON-RPC protocol
- Request: `{jsonrpc: "2.0", method: "tools/call", params: .. .}`
- Response: `{jsonrpc: "2.0", result: ...}`

Error handling:

- Network errors (timeout, connection refused)
- Server errors (tool execution failed)
- Validation errors (invalid parameters)
- Retry logic (exponential backoff)

Timeout configuration:

- Connection timeout (5-10 seconds)
- Request timeout (30-60 seconds dla long operations)
- CancellationToken support

3.2 VS Code MCP Integration

Copilot + MCP

Auto-discovery:

- VS Code Copilot automatycznie wykrywa MCP servers
- Configuration w `. vscode/settings.json`
- Lista servers z paths i args

Chat integration:

- User chat z Copilot
- Copilot analizuje - potrzebuje MCP tool
- Automatyczne wywołanie tool
- Wynik injected do context
- Copilot odpowiada z pełnym kontekstem

Context injection:

- MCP resources dodane do context window
- LLM ma dostęp do local docs, configs
- Better code suggestions (context-aware)

Dokumentacja:

- code.visualstudio.com/docs/copilot/chat-mcp

4 Narzędzia (Tools) w MCP

4.1 Definiowanie Tools

Anatomia Tool

Komponenty:
1. Name (Nazwa):

- Unikalna w ramach servera
- camelCase convention
- Opisowa (searchDatabase, sendEmail, readFile)

2. Description (Opis):

- Jasny, precyzyjny opis funkcjonalności
- LLM używa do decision making (kiedy użyć tool)
- Include use case i constraints
- Przykład: "Search customer database by country. Returns list of customers with name, email, and registration date."

3. Input Parameters (Parametry wejściowe):

- JSON Schema definition
- Required vs optional fields
- Type constraints (string, number, boolean, object)
- Validation rules (min/max, pattern, enum)

4. Output (Wynik):

- Return type description
- Schema dla structured output
- Error cases (co zwraca przy failure)

4.2 Przykładowe Tools

Przykłady Narzędzi

1. **searchDatabase**
   - Description: "Query SQL database"
   - Input: `{sql: string (required), limit: number (optional)}`
   - Output: `List<Row>`
   - Use case: data retrieval, analytics
2. **fetchWeather**
   - Description: "Get current weather for city"
   - Input: `{city: string (required), units: 'metric'|'imperial' (optional)}`
   - Output: `{temp: number, condition: string, humidity: number}`
   - Use case: real-time data
3. **sendEmail**
   - Description: "Send email via SMTP"
   - Input: `{to: string, subject: string, body: string}`
   - Output: `{success: bool, messageId: string}`
   - Use case: notifications, communication
4. **readFile**
   - Description: "Read file content from path"
   - Input: `{path: string}`
   - Output: `{content: string, size: number}`
   - Use case: local file access
5. **executeSQL**
   - Description: "Execute SQL query (SELECT only)"
   - Input: `{query: string, database: string}`
   - Output: `{rows: array, rowCount: number}`
   - Use case: business intelligence queries

4.3 JSON Schema Validation

Walidacja Parametrów

Typy podstawowe:

- `string` - tekst
- `number` - liczby (int lub float)
- `boolean` - true/false
- `object` - zagnieżdżone obiekty
- `array` - listy

Constraints:

- `minLength/maxLength` - dla stringów
- `minimum/maximum` - dla liczb
- `pattern` - regex dla stringów
- `enum` - limited set of values
- `format` - email, uri, date, etc.

Required fields:

- `required: ["field1", "field2"]`
- Walidacja przed wykonaniem tool
- Error jeśli brakuje required field

Przykład schema:

- Type: object
- Properties: `{country: {type: "string", minLength: 2}}`
- Required: `["country"]`

5 Ćwiczenia Praktyczne

5.1 Ćwiczenie 1: Prosty Serwer MCP

Stworzenie Basic Server w C#

Cel: Prosty MCP server z jednym tool
Kroki:

1. Utwórz . NET console app
2. Dodaj MCP SDK (NuGet)
3. Zdefiniuj tool: `getCurrentTime`
4. Description: "Returns current server time"
5. Parameters: `{timezone: string (optional)}`
6. Return: `{time: string, timezone: string}`
7. Configure stdio transport
8. Test w VS Code Copilot

Test:

- VS Code: Add MCP server do settings
- Copilot Chat: "What time is it?"
- Verify tool invocation

5.2 Ćwiczenie 2: Klient MCP

Wywołanie Tool przez LLM

Cel: Custom aplikacja wywołująca MCP server
Kroki:

1. Utwórz HTTP MCP server (z ćwiczenia 1)
2. Deploy lokalnie (localhost:5000)
3. Utwórz console client app
4. Implement discovery (list tools)
5. Implement tool invocation
6. Integrate z OpenAI API (function calling)
7. LLM decyduje kiedy użyć tool
8. Client wywołuje MCP tool
9. LLM przetwarza result

Flow:

- User → OpenAI → decyzja: use tool → MCP Server → result → OpenAI → answer

5.3 Ćwiczenie 3: Dostęp do Bazy Danych

Database Integration

Cel: MCP server z database access
Tools:

1. `queryCustomers` - SELECT customers by filters
2. `getOrderStats` - aggregate statistics
3. `searchProducts` - full-text search

Implementation:

- Entity Framework Core
- SQL Server / SQLite
- Parameterized queries (SQL injection prevention)
- Error handling (connection errors, timeouts)
- Logging (query execution time)

Security:

- Read-only access (SELECT only)
- Query whitelist (allowed tables)
- Row-level security (jeśli applicable)

5.4 Ćwiczenie 4: Deployment na Azure

Production Deployment

Cel: Host MCP server w Azure
Opcje:

1. **Azure App Service**
   - HTTP transport
   - Easy deployment (VS / GitHub Actions)
   - Auto-scaling
2. **Azure Container Apps**
   - Docker container
   - Microservices architecture
   - Serverless scaling
3. **Azure Functions**
   - Serverless (pay-per-execution)
   - HTTP trigger
   - Tool jako function endpoint

Configuration:

- API key w Azure Key Vault
- Connection strings z App Settings
- CORS configuration
- Application Insights (monitoring)

Security:

- HTTPS only
- API key authentication
- Rate limiting (Azure API Management)
- IP whitelisting (optional)

6 Semantic Kernel

Microsoft Semantic Kernel

Alternatywny framework dla AI agents
Czym jest:

- SDK od Microsoft do budowania AI applications
- Integration z OpenAI, Azure OpenAI, Hugging Face
- Plugin system (podobny do MCP tools)
- Planners (multi-step reasoning)

Vs MCP:

- SK = application framework (higher level)
- MCP = communication protocol (lower level)
- SK może używać MCP jako transport layer
- SK ma więcej features (memory, planners)

Use case:

- Complex multi-agent systems
- Enterprise AI applications
- Microsoft ecosystem (Azure OpenAI)

Dokumentacja:

- learn.microsoft.com/semantic-kernel

7 Pytania kontrolne

1. Czym jest Model Context Protocol i jaki problem rozwiązuje?
2. Wymień 3 komponenty architektury MCP.
3. Czym różni się MCP od Function Calling?
4. Co to są Tools, Resources i Prompts w MCP?
5. Kiedy użyć stdio transport a kiedy HTTP?
6. Jakie są zalety i wady HTTP transport?
7. Co to jest JSON Schema i do czego służy w MCP?
8. Jak wygląda workflow wywołania MCP tool przez LLM?
9. Co powinien zawierać dobry opis (description) tool?
10. Jakie są security considerations dla HTTP MCP server?