Introduction to Model Context Protocol (MCP)

The Model Context Protocol (MCP) is a cutting-edge framework designed to standardize interactions between AI models and client applications. MCP serves as a bridge between AI models and the applications that use them, providing a consistent interface regardless of the underlying model implementation.

Key aspects of MCP:

- **Standardized Communication**: MCP establishes a common language for applications to communicate with AI models
- **Enhanced Context Management**: Allows for efficient passing of contextual information to AI models
- **Cross-platform Compatibility**: Works across various programming languages including C#, Java, JavaScript, Python, and TypeScript
- **Seamless Integration**: Enables developers to easily integrate different AI models into their applications

MCP is particularly valuable in AI agent development as it allows agents to interact with various systems and data sources through a unified protocol, making agents more flexible and powerful.

Learning Objectives

- Understand what MCP is and its role in AI agent development
- Set up and configure an MCP server for GitHub integration
- Build a multi-agent system using MCP tools
- Implement RAG (Retrieval Augmented Generation) with Azure Cognitive Search

Prerequisites

- Python 3.8+
- Node.js 14+
- Azure subscription
- GitHub account
- Basic understanding of Semantic Kernel

Setup Instructions

- **Environment Setup**

```
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install -r requirements.txt
```

- **Configure Azure Services**

  - Create an Azure Cognitive Search resource
  - Set up Azure OpenAI service
  - Configure environment variables in `.env`

- **MCP Server Setup**

```
npm install -g @modelcontextprotocol/server-github
```

Project Structure

```
11-mcp/
├── code_samples/
│   └── github-mcp/
│       ├── app.py               # Main application
│       ├── event-descriptions.md  # Event data
│       └── MCP_SETUP.md         # Setup guide
├── README.md
└── requirements.txt
```

Core Components

1. Multi-Agent System

- GitHub Agent: Repository analysis
- Hackathon Agent: Project recommendations
- Events Agent: Tech event suggestions

2. Azure Integration

- Cognitive Search for event indexing
- Azure OpenAI for agent intelligence
- RAG pattern implementation

3. MCP Tools

- GitHub repository analysis
- Code inspection
- Metadata extraction

Code Walkthrough

The sample demonstrates:

1. MCP server integration
2. Multi-agent orchestration
3. Azure Cognitive Search integration
4. RAG pattern implementation

Key features:

- Real-time GitHub repository analysis
- Intelligent project recommendations
- Event matching using Azure Search
- Streaming responses with Chainlit

Running the Sample

For detailed setup instructions and more information, refer to the Github MCP Server Example README.

1. Start the MCP server:

   ```
   npx @modelcontextprotocol/server-github
   ```

Launch the application:

```
chainlit run app.py -w
```

Test the integration:

```
Example query: "Analyze repositories for username: <github_username>"
```

Troubleshooting

Common issues and solutions:

1. MCP Connection Issues
    - Verify server is running
    - Check port availability
    - Confirm GitHub tokens
2. Azure Search Issues
    - Validate connection strings
    - Check index existence
    - Verify document upload

Next Steps

- Explore additional MCP tools
- Implement custom agents
- Enhance RAG capabilities
- Add more event sources