/mcp-builder

Source: `~/.claude/skills/mcp-builder/SKILL.md`

---

name: mcp-builder description: Guide for creating high-quality MCP (Model Context Protocol) servers that enable LLMs to interact with external services through well-designed tools. Use when building MCP servers to integrate external APIs or services, whether in Python (FastMCP) or Node/TypeScript (MCP SDK). license: Complete terms in LICENSE.txt

MCP Server Development Guide

Overview

Create MCP (Model Context Protocol) servers that enable LLMs to interact with external services through well-designed tools. The quality of an MCP server is measured by how well it enables LLMs to accomplish real-world tasks.

Process

? High-Level Workflow

Creating a high-quality MCP server involves four main phases:

Phase 1: Deep Research and Planning

1.1 Understand Modern MCP Design

API Coverage vs. Workflow Tools: Balance comprehensive API endpoint coverage with specialized workflow tools. Workflow tools can be more convenient for specific tasks, while comprehensive coverage gives agents flexibility to compose operations. Performance varies by client—some clients benefit from code execution that combines basic tools, while others work better with higher-level workflows. When uncertain, prioritize comprehensive API coverage.

Tool Naming and Discoverability: Clear, descriptive tool names help agents find the right tools quickly. Use consistent prefixes (e.g., `github_create_issue`, `github_list_repos`) and action-oriented naming.

Context Management: Agents benefit from concise tool descriptions and the ability to filter/paginate results. Design tools that return focused, relevant data. Some clients support code execution which can help agents filter and process data efficiently.

Actionable Error Messages: Error messages should guide agents toward solutions with specific suggestions and next steps.

1.2 Study MCP Protocol Documentation

Navigate the MCP specification:

Start with the sitemap to find relevant pages: `https://modelcontextprotocol.io/sitemap.xml`

Then fetch specific pages with `.md` suffix for markdown format (e.g., `https://modelcontextprotocol.io/specification/draft.md`).

Key pages to review:

- Specification overview and architecture
- Transport mechanisms (streamable HTTP, stdio)
- Tool, resource, and prompt definitions

1.3 Study Framework Documentation

Recommended stack:

- **Language**: TypeScript (high-quality SDK support and good compatibility in many execution environments e.g. MCPB. Plus AI models are good at generating TypeScript code, benefiting from its broad usage, static typing and good linting tools)
- **Transport**: Streamable HTTP for remote servers, using stateless JSON (simpler to scale and maintain, as opposed to stateful sessions and streaming responses). stdio for local servers.

Load framework documentation:

- **MCP Best Practices**: □□ View Best Practices - Core guidelines

For TypeScript (recommended):

- **TypeScript SDK**: Use WebFetch to load `https://raw.githubusercontent.com/modelcontextprotocol/typescript-sdk/main/README.md`
- ⚡ TypeScript Guide - TypeScript patterns and examples

For Python:

- **Python SDK**: Use WebFetch to load `https://raw.githubusercontent.com/modelcontextprotocol/python-sdk/main/README.md`
- □□ Python Guide - Python patterns and examples

1.4 Plan Your Implementation

Understand the API: Review the service's API documentation to identify key endpoints, authentication requirements, and data models. Use web search and WebFetch as needed.

Tool Selection: Prioritize comprehensive API coverage. List endpoints to implement, starting with the most common operations.

Phase 2: Implementation

2.1 Set Up Project Structure

See language-specific guides for project setup:

- ⚡ TypeScript Guide - Project structure, package.json, tsconfig.json
- 🐍 Python Guide - Module organization, dependencies

2.2 Implement Core Infrastructure

Create shared utilities:

- API client with authentication
- Error handling helpers
- Response formatting (JSON/Markdown)
- Pagination support

2.3 Implement Tools

For each tool:

Input Schema:

- Use Zod (TypeScript) or Pydantic (Python)
- Include constraints and clear descriptions
- Add examples in field descriptions

Output Schema:

- Define `outputSchema` where possible for structured data
- Use `structuredContent` in tool responses (TypeScript SDK feature)
- Helps clients understand and process tool outputs

Tool Description:

- Concise summary of functionality
- Parameter descriptions
- Return type schema

Implementation:

- Async/await for I/O operations
- Proper error handling with actionable messages
- Support pagination where applicable
- Return both text content and structured data when using modern SDKs

Annotations:

- `readOnlyHint`: true/false
- `destructiveHint`: true/false
- `idempotentHint`: true/false
- `openWorldHint`: true/false

Phase 3: Review and Test

3.1 Code Quality

Review for:

- No duplicated code (DRY principle)
- Consistent error handling
- Full type coverage
- Clear tool descriptions

3.2 Build and Test

TypeScript:

- Run `npm run build` to verify compilation
- Test with MCP Inspector: `npx @modelcontextprotocol/inspector`

Python:

- Verify syntax: `python -m py_compile your_server.py`
- Test with MCP Inspector

See language-specific guides for detailed testing approaches and quality checklists.

Phase 4: Create Evaluations

After implementing your MCP server, create comprehensive evaluations to test its effectiveness.

Load Evaluation Guide for complete evaluation guidelines.

4.1 Understand Evaluation Purpose

Use evaluations to test whether LLMs can effectively use your MCP server to answer realistic, complex questions.

4.2 Create 10 Evaluation Questions

To create effective evaluations, follow the process outlined in the evaluation guide:

1. **Tool Inspection**: List available tools and understand their capabilities
2. **Content Exploration**: Use READ-ONLY operations to explore available data
3. **Question Generation**: Create 10 complex, realistic questions
4. **Answer Verification**: Solve each question yourself to verify answers

4.3 Evaluation Requirements

Ensure each question is:

- **Independent**: Not dependent on other questions
- **Read-only**: Only non-destructive operations required
- **Complex**: Requiring multiple tool calls and deep exploration
- **Realistic**: Based on real use cases humans would care about
- **Verifiable**: Single, clear answer that can be verified by string comparison
- **Stable**: Answer won't change over time

4.4 Output Format

Create an XML file with this structure:

```
<evaluation>
  <qa_pair>
    <question>Find discussions about AI model launches with animal codenames. One model needed a specific safety designation that uses the format ASL-X. What number X was being determined for the model named after a spotted wild cat?</question>
    <answer>3</answer>
  </qa_pair>
<!-- More qa_pairs... -->
</evaluation>
```

Reference Files

? Documentation Library

Load these resources as needed during development:

Core MCP Documentation (Load First)

- **MCP Protocol**: Start with sitemap at `https://modelcontextprotocol.io/sitemap.xml`, then fetch specific pages with `.md` suffix
- MCP Best Practices - Universal MCP guidelines including:
  - Server and tool naming conventions
  - Response format guidelines (JSON vs Markdown)
  - Pagination best practices

    - Transport selection (streamable HTTP vs stdio)
    - Security and error handling standards

SDK Documentation (Load During Phase 1/2)

- **Python SDK**: Fetch from `https://raw.githubusercontent.com/modelcontextprotocol/python-sdk/main/README.md`
- **TypeScript SDK**: Fetch from `https://raw.githubusercontent.com/modelcontextprotocol/typescript-sdk/main/README.md`

Language-Specific Implementation Guides (Load During Phase 2)

- Python Implementation Guide - Complete Python/FastMCP guide with:
    - Server initialization patterns
    - Pydantic model examples
    - Tool registration with `@mcp.tool`
    - Complete working examples
    - Quality checklist
- ⚡ TypeScript Implementation Guide - Complete TypeScript guide with:
    - Project structure
    - Zod schema patterns
    - Tool registration with `server.registerTool`
    - Complete working examples
    - Quality checklist

Evaluation Guide (Load During Phase 4)

- Evaluation Guide - Complete evaluation creation guide with:
    - Question creation guidelines
    - Answer verification strategies
    - XML format specifications
    - Example questions and answers
    - Running an evaluation with the provided scripts

Revision #5
Created 2026-02-18 08:42:04 UTC by John
Updated 2026-06-21 20:01:33 UTC by John