MCP Server Python Project Structure

Python MCP server using FastMCP. Expose tools, resources, and prompts to AI assistants with type-safe decorators.

#mcp #python #ai #tools #claude #fastmcp #model-context-protocol

PNG PDF Copy Prompt

Project Directory

```
mcp-server/
  src/ Server source c…
    __init__.py
    server.py FastMCP server …
    tools/ Tool definitions
      __init__.py
      search.py Example tool
      database.py
    resources/ Resource provid…
      __init__.py
      files.py File system res…
      config.py
    prompts/ Prompt templates
      __init__.py
      analysis.py
    utils/ Shared utilities
      __init__.py
      http.py HTTP client hel…
      validators.py
  tests/
    __init__.py
    conftest.py Pytest fixtures
    test_tools.py
    test_resources.py
  pyproject.toml Project config,…
  uv.lock Dependency lock…
  .env.example Environment tem…
  .gitignore
  README.md
```

Why This Structure?

MCP (Model Context Protocol) lets you extend AI assistants with custom capabilities. This structure separates tools, resources, and prompts into distinct modules. FastMCP uses decorators and type hints to auto-generate protocol definitions.

Key Directories

src/server.py - FastMCP instance, mounts all capabilities

src/tools/ - Functions the AI can call (actions)

src/resources/ - Data the AI can read (context)

src/prompts/ - Reusable prompt templates

Getting Started

1. `uv init mcp-server && cd mcp-server`
2. `uv add "mcp[cli]" httpx`
3. `uv run python -m src.server`
4. `claude mcp add my-server ./run.sh`

FastMCP Basics

```python
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("my-server")

@mcp.tool()
async def search(query: str) -> str:
    """Search for information."""
    # Implementation here
    return f"Results for: {query}"

@mcp.resource("config://settings")
def get_settings() -> str:
    """Return current settings."""
    return '{"theme": "dark"}'
```

Transport Options

stdio - Default for local CLI usage (Claude Code)

streamable-http - For remote servers accessible via HTTP

sse - Server-sent events for web integrations

When To Use This

- Adding custom tools to Claude Code or other MCP clients
- Giving AI access to internal APIs or databases
- Creating domain-specific AI capabilities
- Building reusable prompt templates
- Exposing file system or external data to AI

Best Practices

- Never use `print()` with stdio transport—use `logging` to stderr
- Type hints and docstrings auto-generate tool descriptions
- Keep tools focused—one action per function
- Validate inputs before external API calls
- Handle errors gracefully with clear messages

Trade-offs

Protocol overhead - MCP adds complexity vs direct function calls

Debugging - JSON-RPC over stdio harder to debug than REST

Client support - Limited to MCP-compatible AI clients

Testing Strategy

tests/test_tools.py - Unit test tool functions directly

conftest.py - Mock external APIs, provide test fixtures

mcp dev - Use MCP Inspector for interactive testing