

BUILDING AND DEPLOYING AI AGENTS

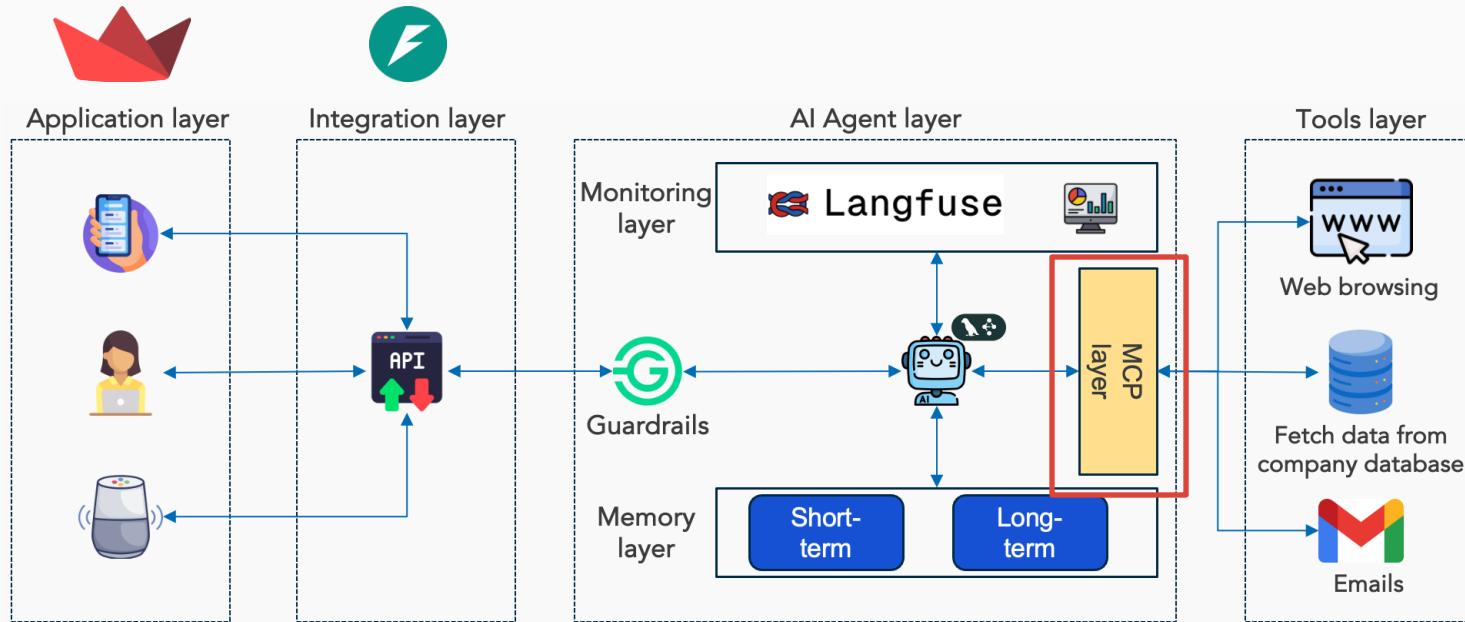
Class 7 - AI Standard Protocols & real-world AI Agent

1. Introducing MCP protocol – deep dive into the most popular AI standard protocol
2. Demo – Building your first MCP and adding it into an Agent
3. AI Agents use cases – Overview
4. Real-world use case experience – Deep dive into a real AI Agent
5. Lessons learned – Sharing real experience
6. Demo – Capturing everything you learned and develop a Travel Agent
7. Reflection - Connecting the dots of what we learned today.
8. M6 assignment – What is the challenge?
9. Wrap up - Next steps

Connecting all dots

Today we will look at the last piece of the puzzle

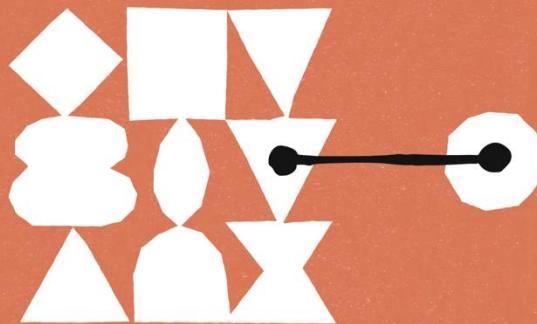
TΛ!



The HTTP moment of AI?

Introducing the Model Context Protocol

25 Nov 2024 • 3 min read

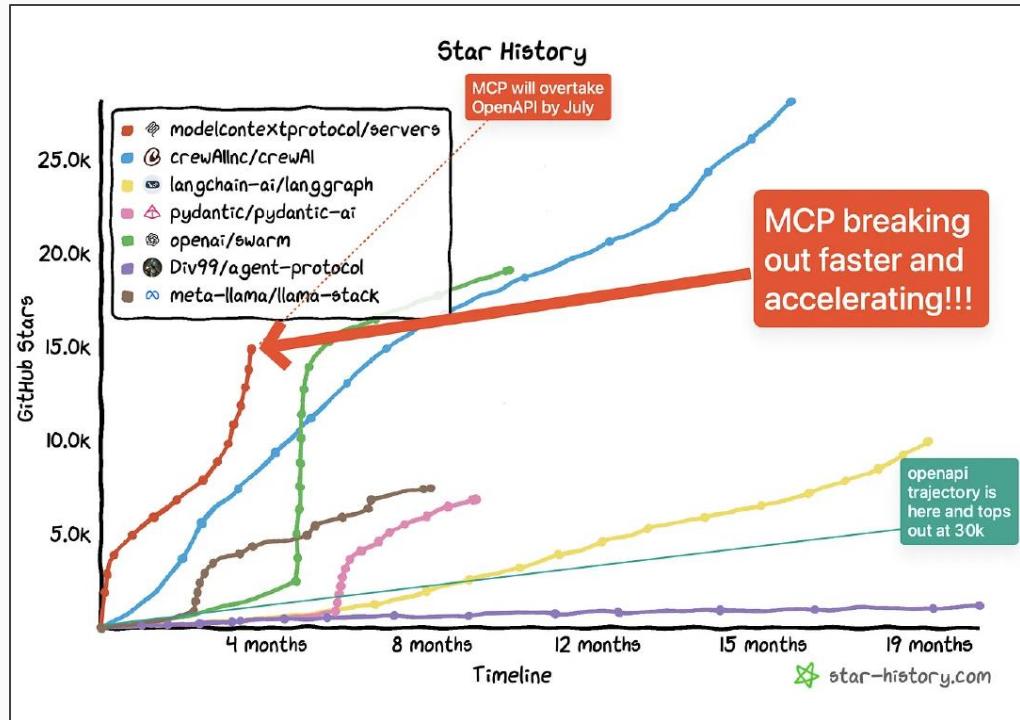


Source: <https://www.anthropic.com/news/model-context-protocol>

MCP is Growing Super Fast

T^A

All major players have already adopted it



Source: <https://medium.com/@manuktiwary/the-model-context-protocol-mcp-a-complete-guide-84ecb9cbaf03>

MCP is an open protocol that enables seamless integration between **AI apps & agents** and your **tools & data sources**.

APIs

Standardize how **web applications** interact with the **backend**:

- Servers
- Databases
- Services

LSP

Standardizes how IDEs interact with **language-specific tools**:

- Code navigation
- Code analysis
- Code intelligence

MCP

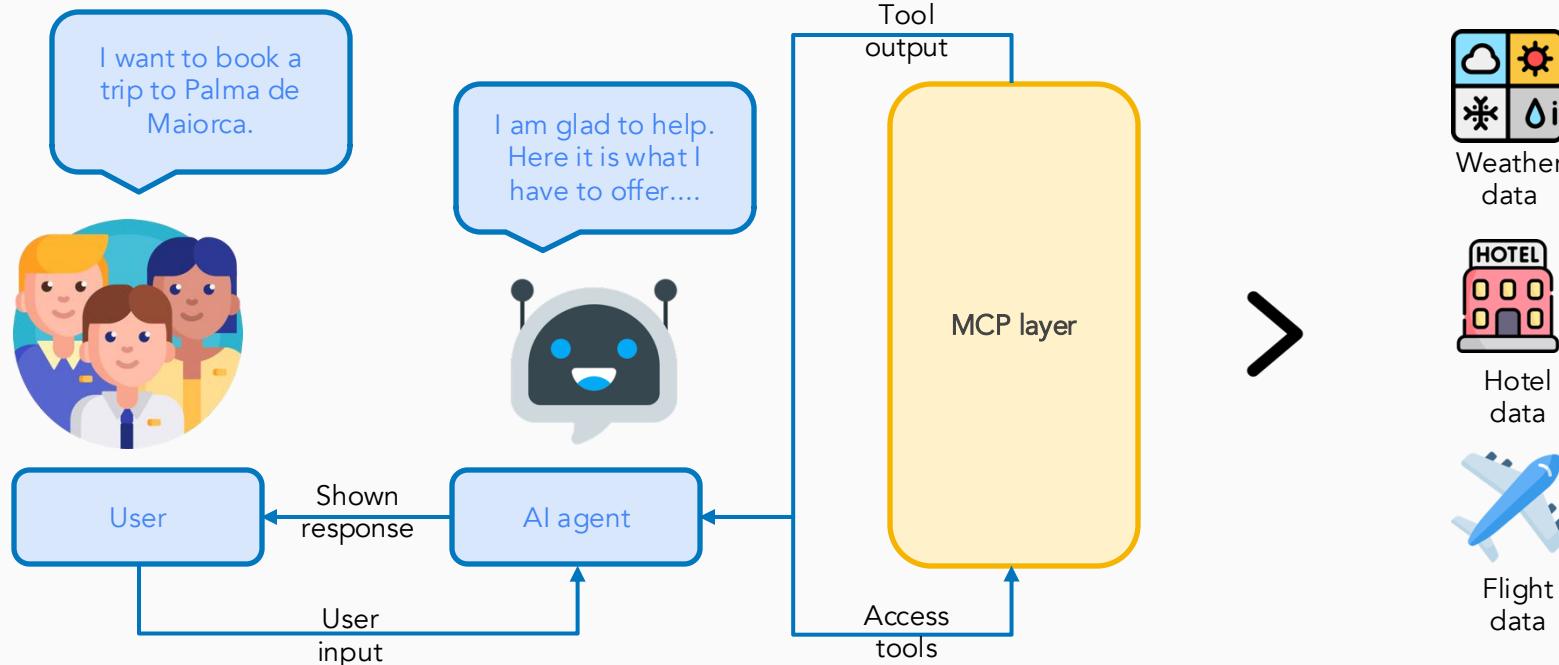
Standardizes how **AI applications** interact with **external systems**:

- Prompts
- Tools
- Resources

The problem MCP solves

TΛ'

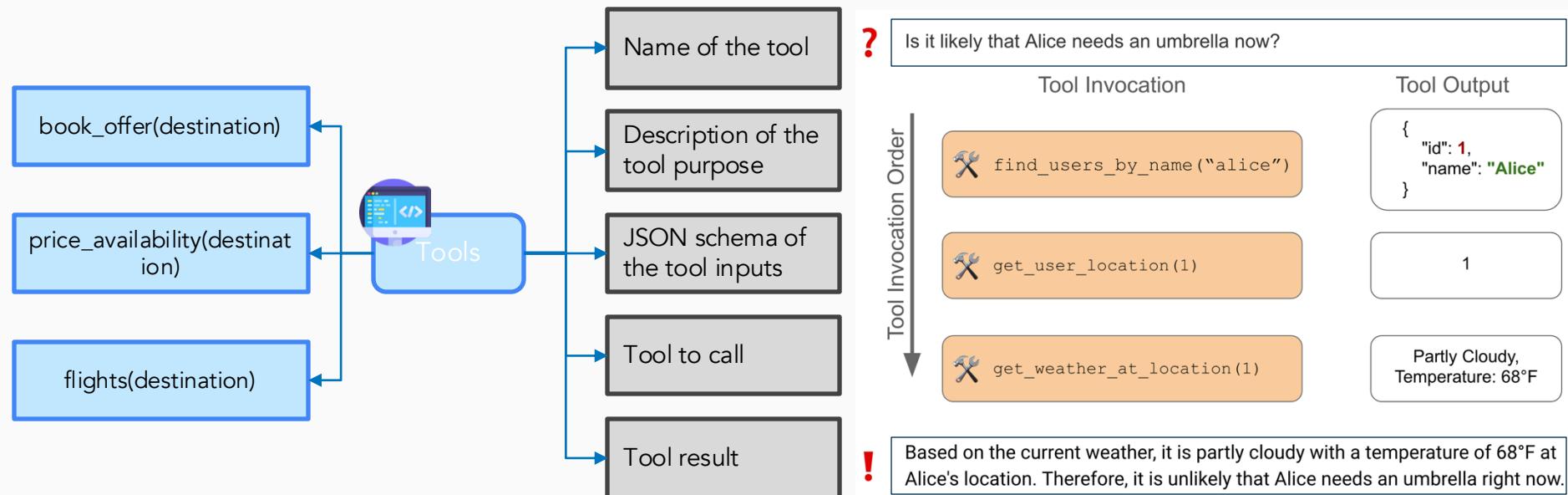
Let's say you're building a chat interface where users can ask a travel assistant about their next trip. A user might ask, "I want to book a trip to Palma de Maiorca?" To handle this, the Agent needs tools to access API's.



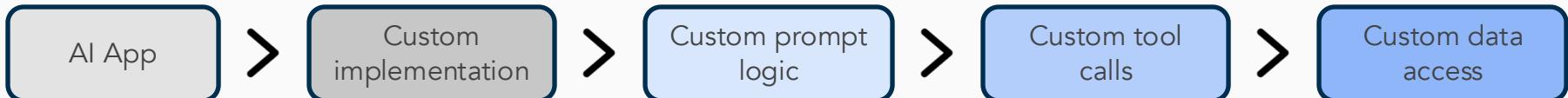
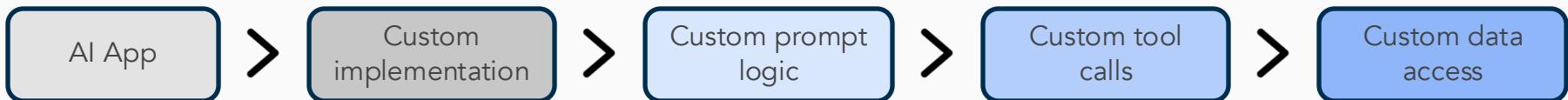
The problem MCP solves

TΛ!

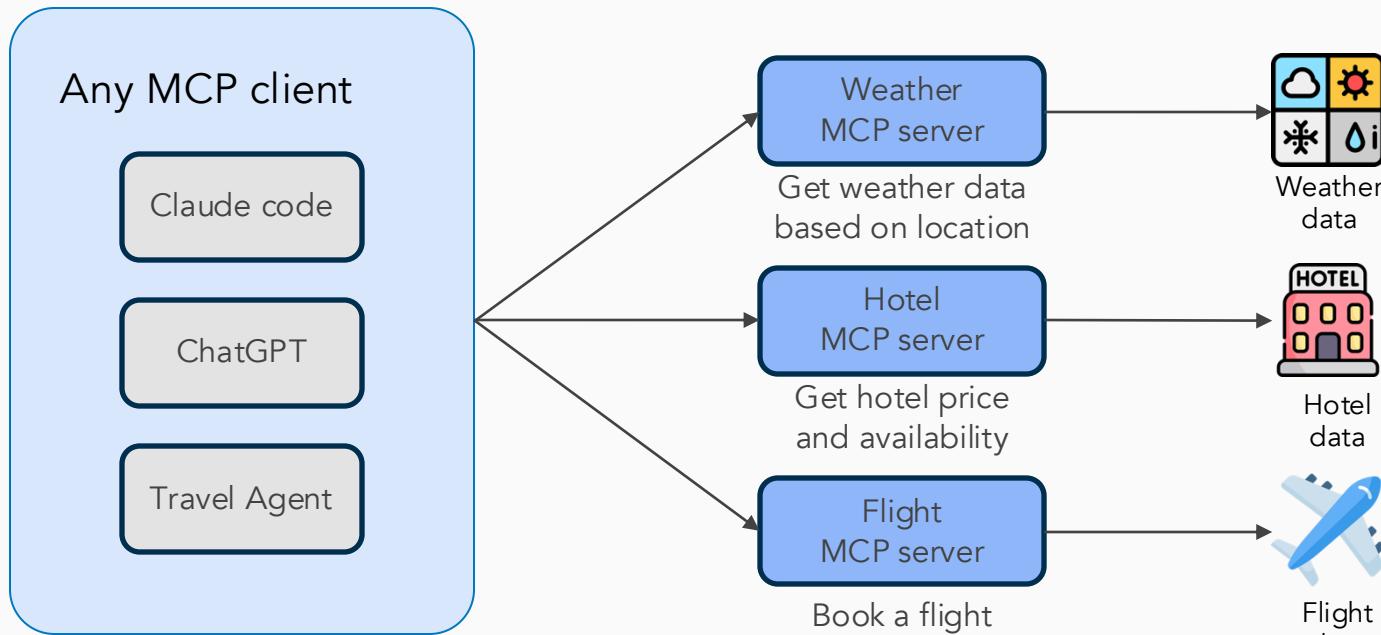
API's have multiple purposes like search accommodations, check price and availability, book and tons more. Without MCP, you'd need to create an incredible number of tool schemas and functions to handle all of API's features.



Fragmented AI development



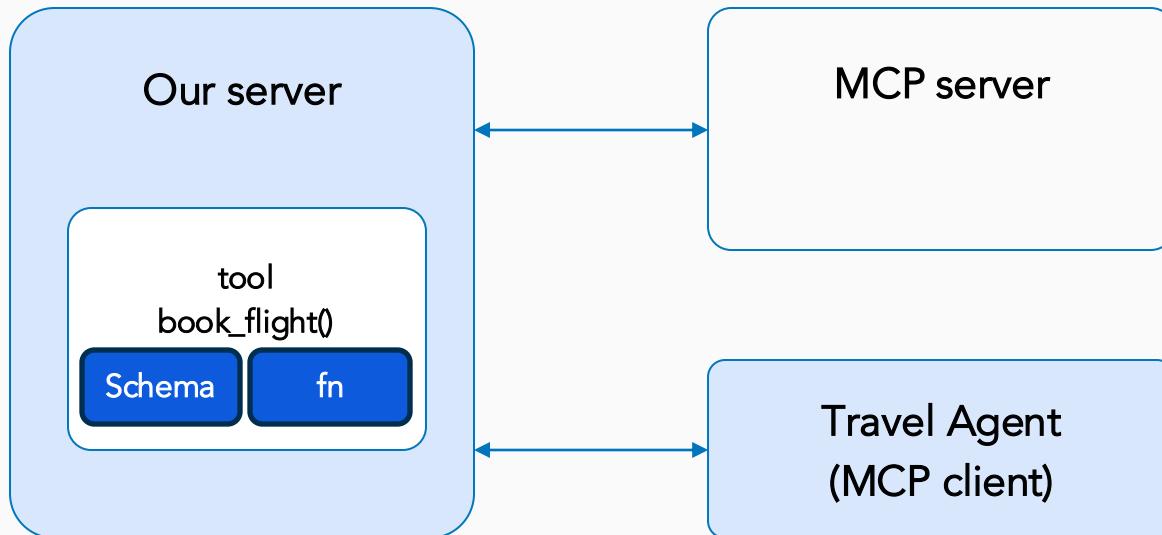
Standardised AI development



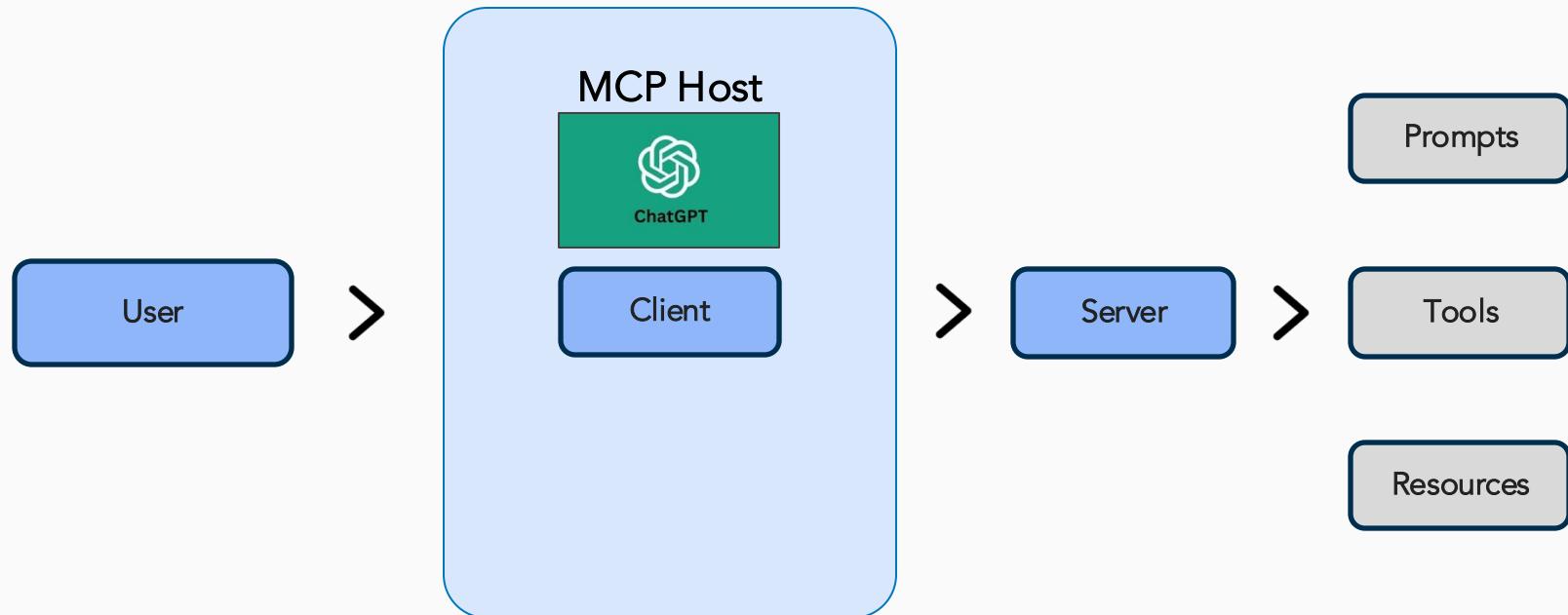
How does MCP work?

TΛ!

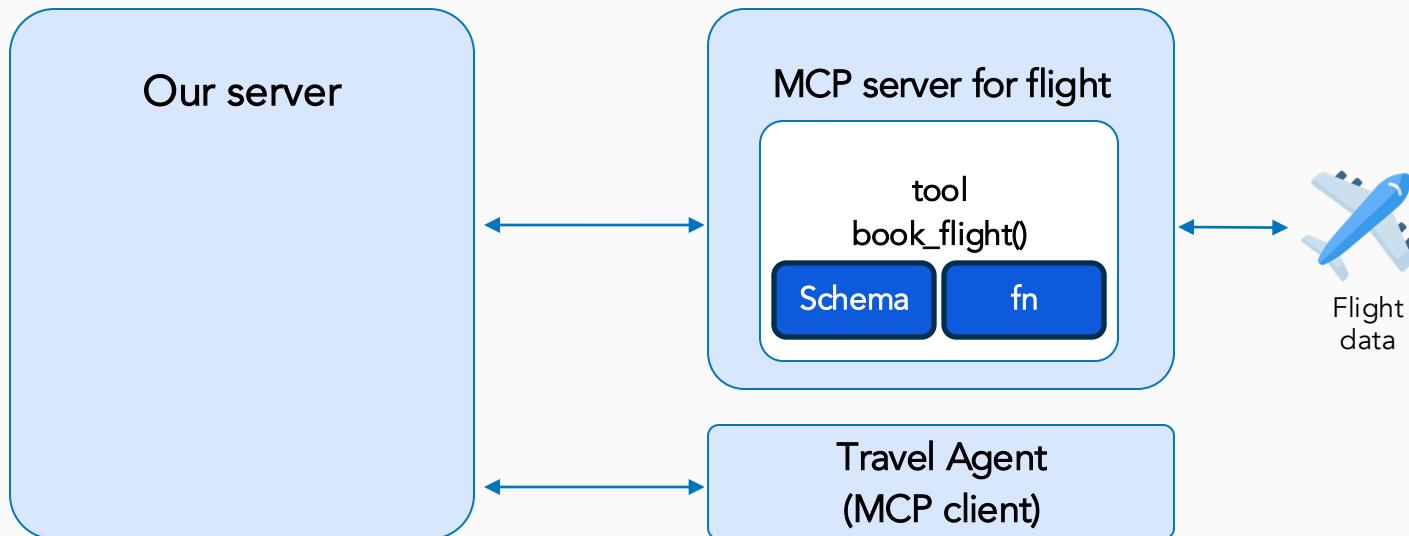
MCP alleviates this burden by transferring tool definitions and execution from your server to dedicated MCP servers. Instead of creating all tools yourself, a dedicated MCP server will manage specific flows, such as search.



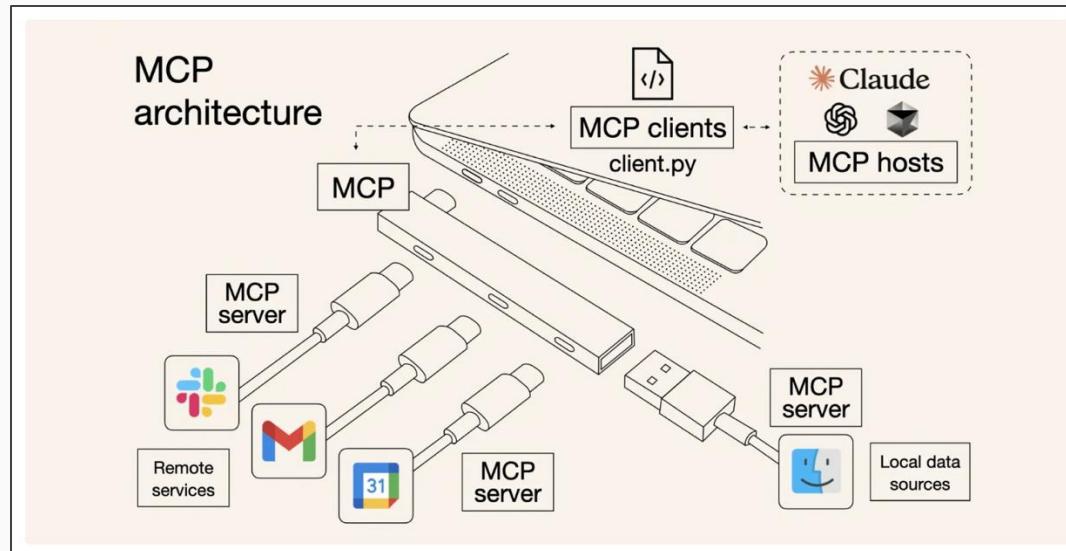
MCP is what makes AI useful for real apps



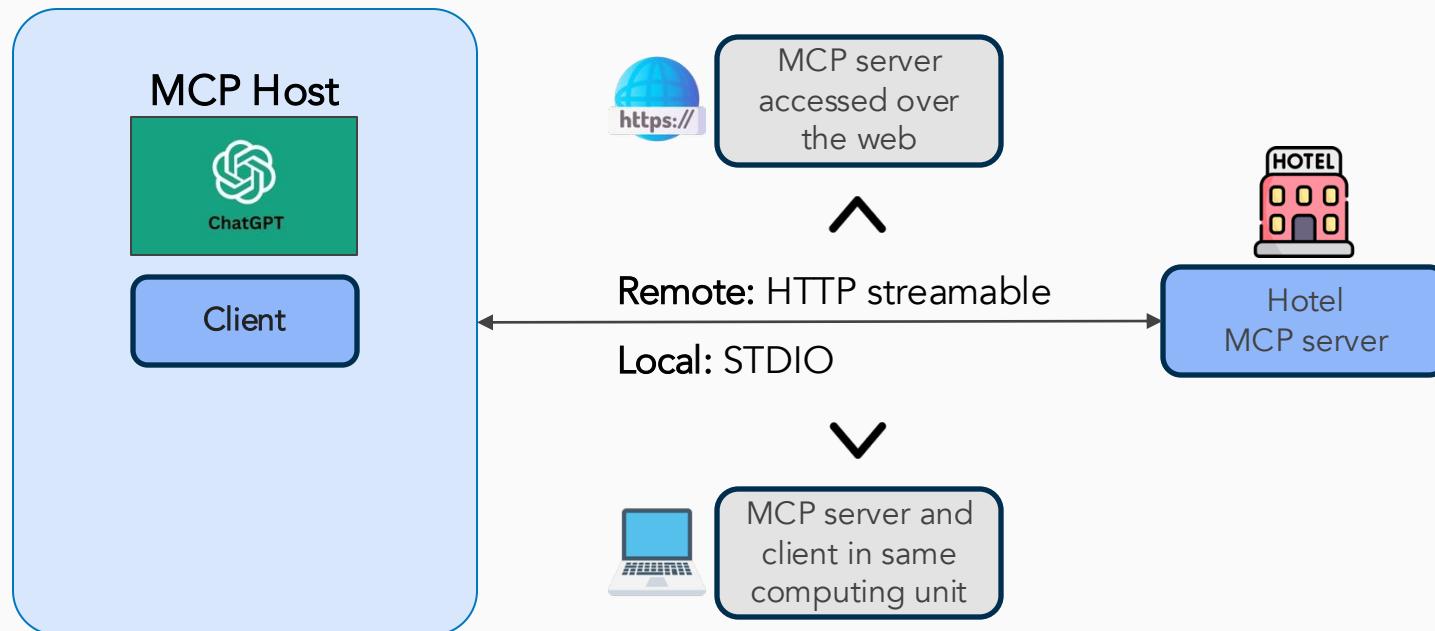
MCP Servers provide access to data or functionality implemented by outside services. They act as specialised interfaces that standardise the exposure of tools, prompts, and resources.



The MCP client serves as the communication bridge between your server and MCP servers. It's your access point to all the tools that an MCP server provides, handling the message exchange and protocol details so your application doesn't have to.



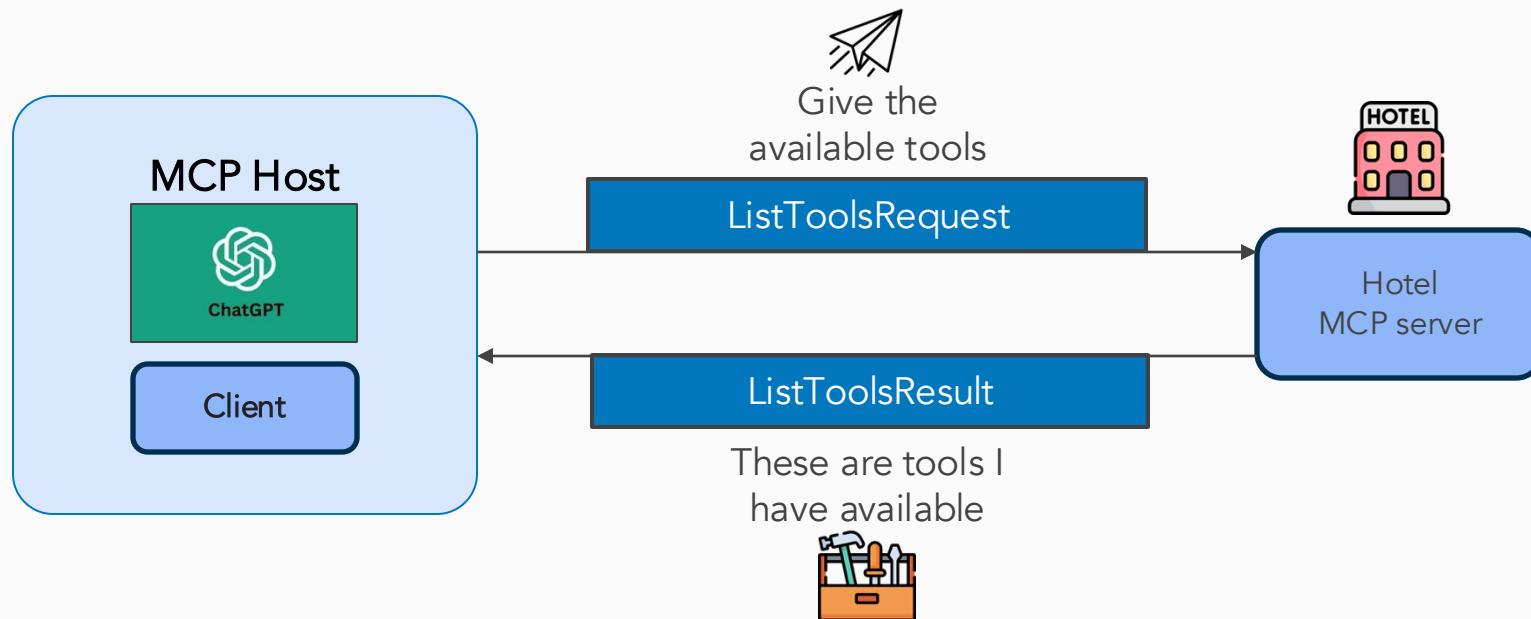
One of MCP's key strengths is being transport agnostic, a fancy way of saying the client and server can communicate over different protocols



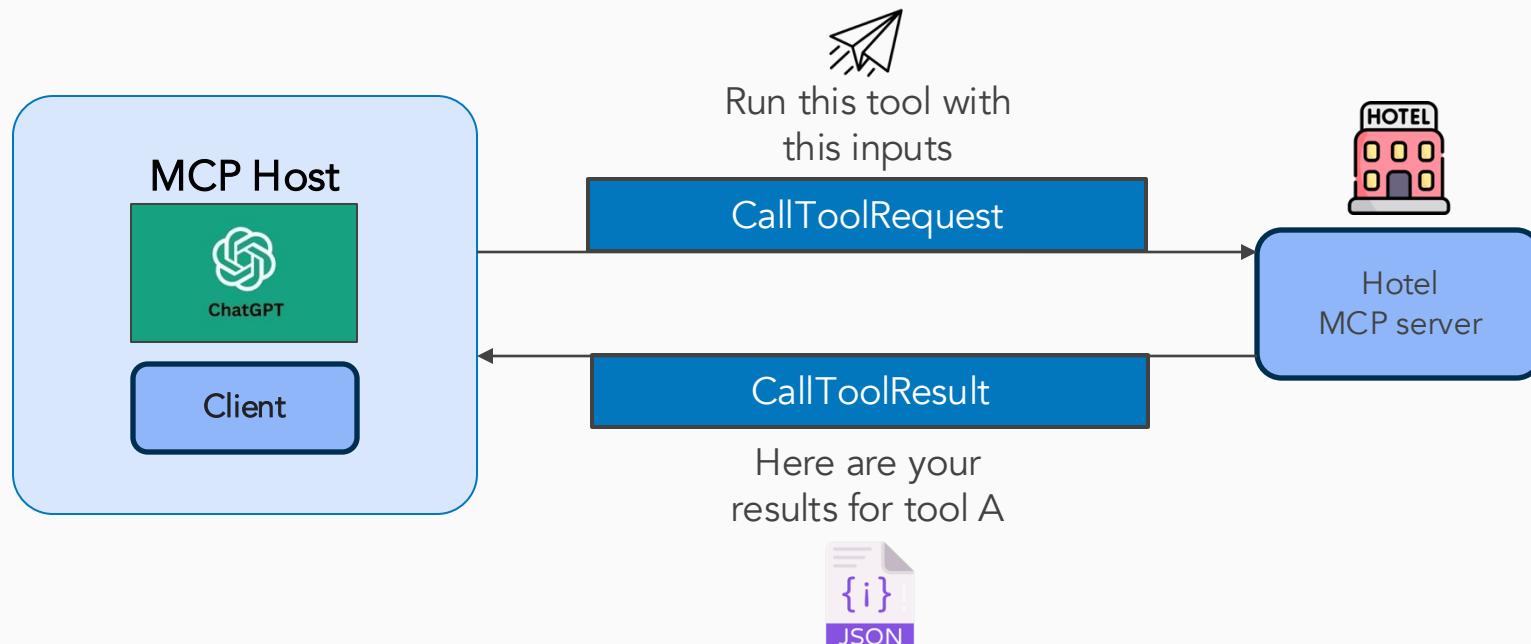
Once connected, the client and server exchange specific message types defined in the MCP specification. The main ones you'll work with are:

- **ListToolsRequest/ListToolsResult:** The client asks the server, "What tools do you provide?" and gets back a list of available tools.
- **CallToolRequest/CallToolResult:** The client asks the server to run a specific tool with given arguments, then receives the results.

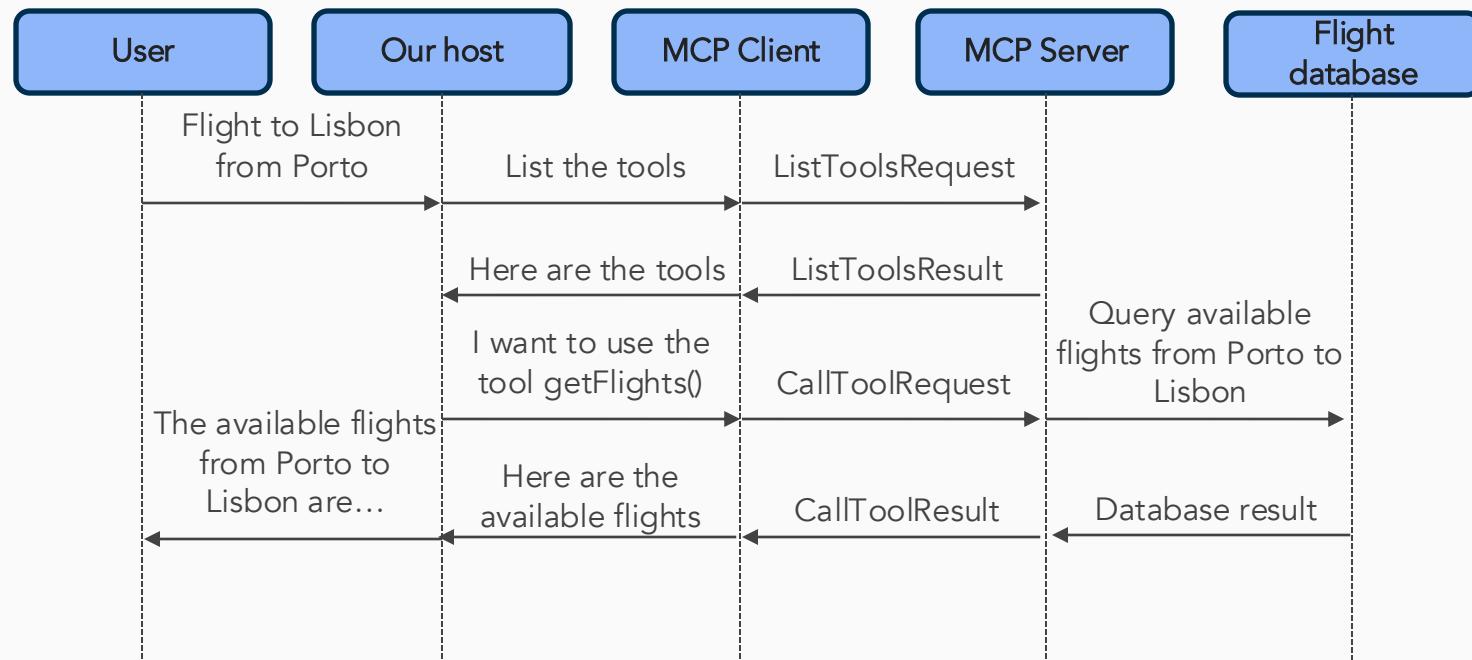
The client can initiate the MCP server connection by requesting the tools and the server responds back with the available tools.



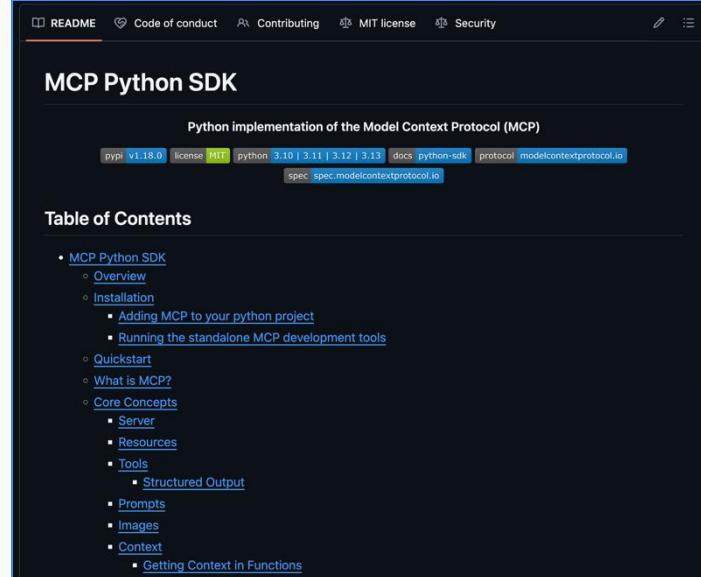
The client can also request to trigger a given tool and receive back its result from the server.



The MCP client abstracts away the complexity of server communication, letting you focus on your application logic while still getting access to powerful external tools.



Building an MCP server becomes much simpler when you use the official Python SDK. Instead of writing complex JSON schemas by hand, you can define tools with decorators and let the SDK handle the heavy lifting.



Source: <https://github.com/modelcontextprotocol/python-sdk>

Let's say you want to build a document mention feature where users can type @document_name to reference file. This requires two operations:

- Getting a list of all available documents (for autocomplete)
- Fetching the contents of a specific document (when mentioned)

```
>_ OpenAI Codex (v0.47.0)
model: gpt-5-codex /model to change
directory: ~/.../buidling-deploying-agents-applications

To get started, describe a task or try one of these commands:
/init - create an AGENTS.md file with instructions for Codex
/status - show current session configuration
/approvals - choose what Codex can do without approval
/model - choose what model and reasoning effort to use
/review - review any changes and find issues

> @classes
classes/class-06-guardrails-ethics/demos/ai-agent-no-guardrails/.env.example
classes/class-06-guardrails-ethics/demos/ai-agent-no-guardrails/README.md
classes/class-06-guardrails-ethics/demos/ai-agent-no-guardrails/main.py
classes/class-06-guardrails-ethics/demos/ai-agent-no-guardrails/requirements.txt
classes/class-06-guardrails-ethics/demos/ai-agent-with-guardrails/.env.example
classes/class-06-guardrails-ethics/demos/ai-agent-with-guardrails/README.md
classes/class-06-guardrails-ethics/demos/ai-agent-with-guardrails/main.py
classes/class-06-guardrails-ethics/demos/ai-agent-with-guardrails/requirements.txt
```

Codex usage example of the decorator @

When a user mentions a document, your system automatically injects the document's contents into the prompt sent to the Client, eliminating the need for the Client to use tools to fetch the information.

```
> OpenAI Codex (v0.47.0)

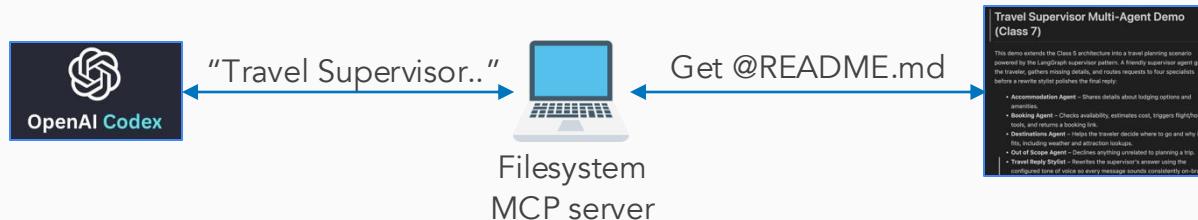
model: gpt-5-codex /model to change
directory: ~/.../building-deploying-agents-applications

To get started, describe a task or try one of these commands:

/init - create an AGENTS.md file with instructions for Codex
/status - show current session configuration
/approvals - choose what Codex can do without approval
/model - choose what model and reasoning effort to use
/review - review any changes and find issues

> Summarise the classes/class-07-industry-case-studies/demos/travel-supervisor-agent/README.md.

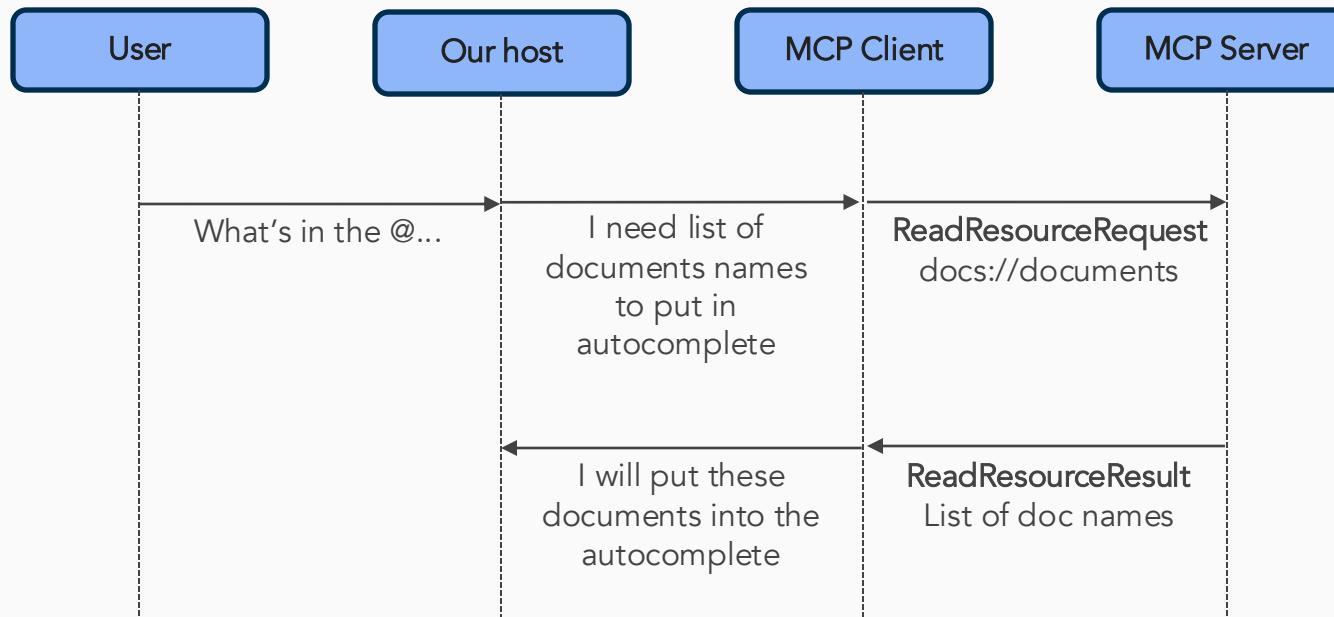
100% context left
```



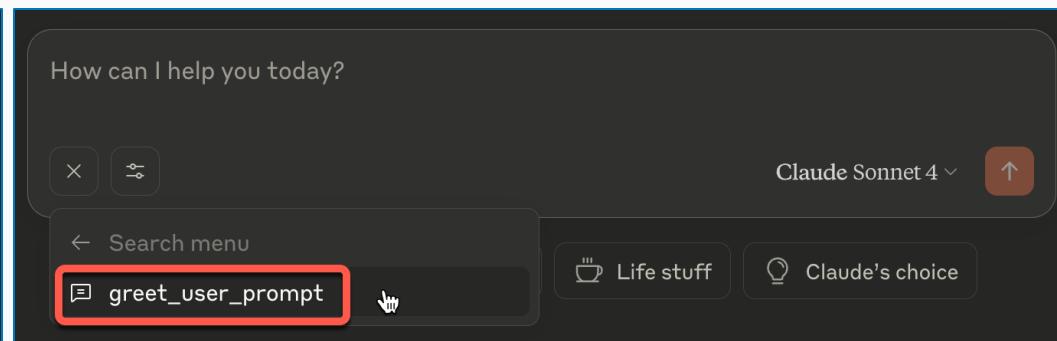
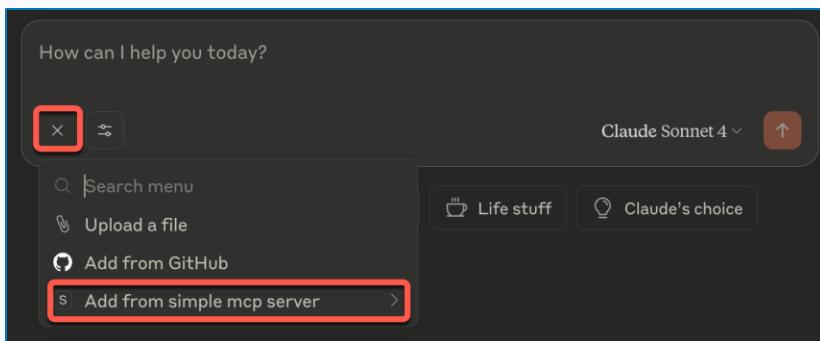
How resources work?

TΛ!

Resources follow a request-response pattern. When your client needs data, it sends a `ReadResourceRequest` with a URI to identify which resource it wants. The MCP server processes this request and returns the data in a `ReadResourceResult`.



Prompts in MCP servers let you define pre-built, high-quality instructions that clients can use instead of writing their own prompts from scratch. Think of them as carefully crafted templates that give better results than what users might come up with on their own.



Source: <https://medium.com/@laurentkubaski/mcp-prompts-explained-including-how-to-actually-use-them-9db13d69d7e2>

Why use Prompts in MCP servers?

TΛ!

As the MCP server author, you can spend time crafting, testing, and evaluating prompts that work consistently across different scenarios. Users benefit from this expertise without having to become prompt engineering experts themselves.

Let's use codex as an example
without prompt templates

Create a README.md for the Travel Agent.

With this prompt Codex will make a lot of assumptions for the README.md structure and content.

Use prompt templates If we want a particular style for our README.md

Please analyze the following project files and create a comprehensive README.md file that includes:

1. Project Title and Description: A clear, concise overview of what the project does
2. Features: Key functionality and capabilities
3. Installation: Step-by-step setup instructions
4. Usage: Examples of how to use the project
5. Configuration: Any environment variables or config files needed
6. Dependencies: Required libraries, frameworks, or tools
7. Project Structure: Brief overview of the codebase organization

Please make the README professional, well-formatted with proper markdown, and suitable for a public repository.

The Python [FastMCP](#) SDK makes server creation straightforward. You can initialise a server with just one line.

```
example.py

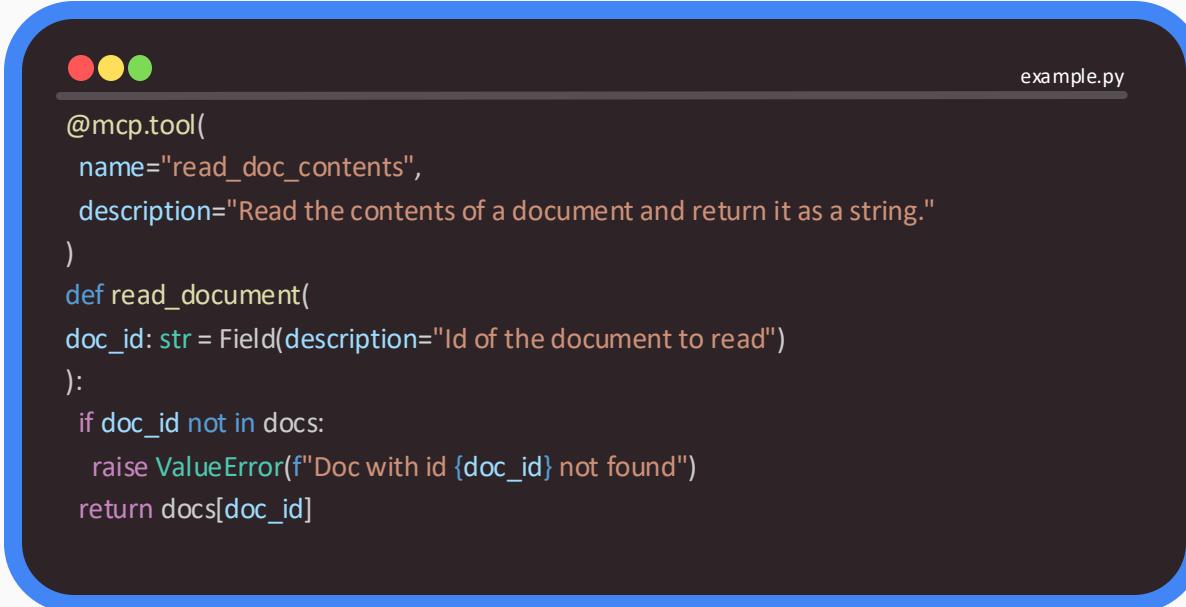
from datetime import datetime
from mcp.server.fastmcp import FastMCP
import logging

mcp = FastMCP("basic-demo")

@mcp.tool()
def get_current_time() -> str:
    return datetime.now().isoformat()

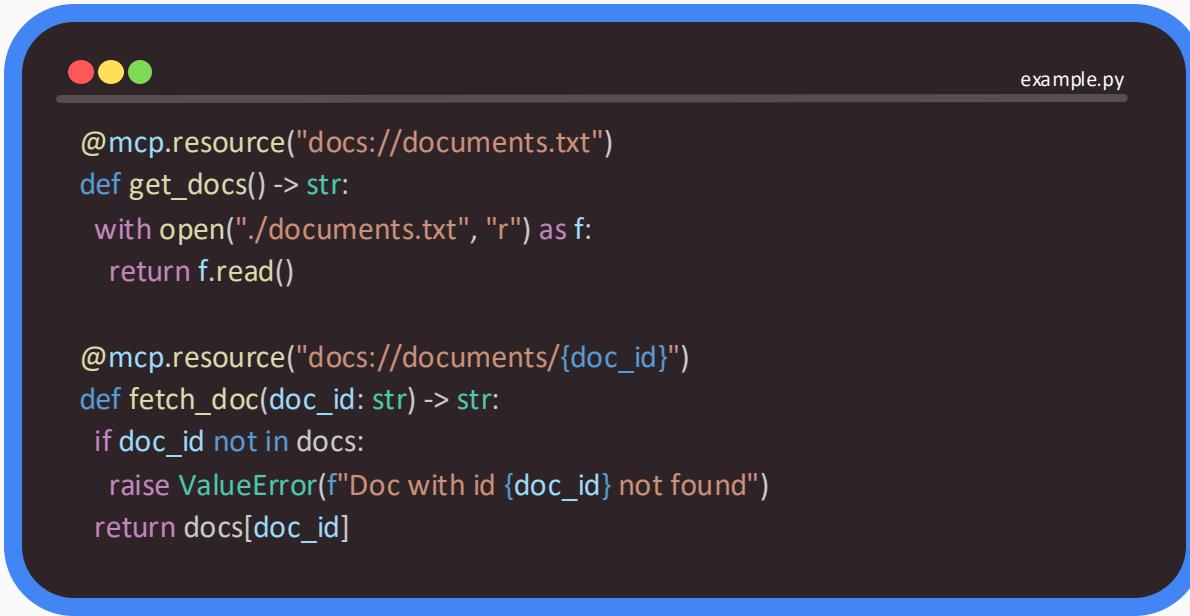
if __name__ == "__main__":
    logging.info("🚀 Basic MCP (stdio)")
    mcp.run(transport="stdio")
```

The SDK uses decorators to define tools. Instead of writing JSON schemas manually, you can use Python type hints and field descriptions. The SDK automatically generates the proper schema that Claude can understand.

A screenshot of a code editor window titled "example.py". The window has a dark theme with a blue border. Inside, there is Python code defining a tool named "read_doc_contents".

```
@mcp.tool(
    name="read_doc_contents",
    description="Read the contents of a document and return it as a string."
)
def read_document(
    doc_id: str = Field(description="Id of the document to read")
):
    if doc_id not in docs:
        raise ValueError(f"Doc with id {doc_id} not found")
    return docs[doc_id]
```

For resources we use the `@mcp.resource` decorator either for direct resources or templated resources.

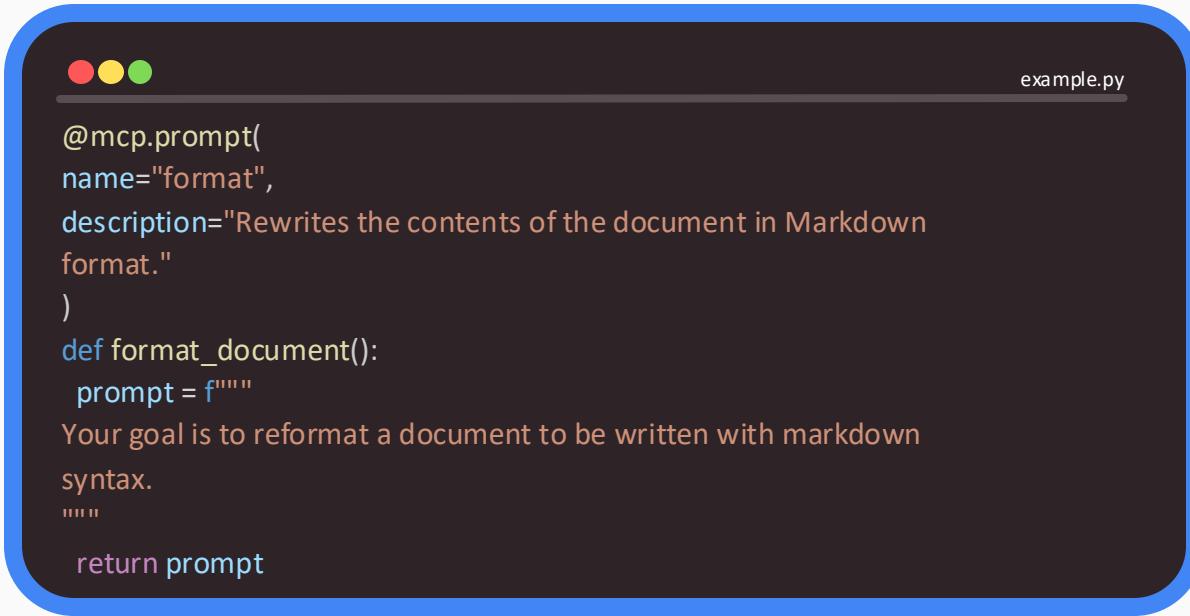


The screenshot shows a code editor window with a dark theme. At the top, there are three colored circular icons (red, yellow, green) followed by the file name "example.py". The code itself is written in Python and uses the `@mcp.resource` decorator from the `mcp` library to define resources. It includes two functions: `get_docs()` for a direct resource and `fetch_doc(doc_id)` for a templated resource that handles document IDs.

```
@mcp.resource("docs://documents.txt")
def get_docs() -> str:
    with open("./documents.txt", "r") as f:
        return f.read()

@mcp.resource("docs://documents/{doc_id}")
def fetch_doc(doc_id: str) -> str:
    if doc_id not in docs:
        raise ValueError(f"Doc with id {doc_id} not found")
    return docs[doc_id]
```

Prompts use a similar decorator pattern to tools and resources:



The screenshot shows a code editor window with a dark theme. At the top, there are three colored circular icons (red, yellow, green) followed by the file name "example.py". The code itself is as follows:

```
@mcp.prompt(  
    name="format",  
    description="Rewrites the contents of the document in Markdown  
    format."  
)  
def format_document():  
    prompt = f"""\n        Your goal is to reformat a document to be written with markdown  
        syntax.  
        """  
  
    return prompt
```

- No manual JSON schema writing required
- Type hints provide automatic validation
- Clear parameter descriptions help Claude understand tool usage
- Error handling integrates naturally with Python exceptions
- Tool registration happens automatically through decorators
- The MCP Python SDK transforms tool creation from a complex schema-writing exercise into simple Python function definitions. This approach makes it much easier to build and maintain MCP servers while ensuring Claude receives properly formatted tool specifications.

Demo – Let's build a simple MCP

And inspect it



Building our first MCP server

TΛ!

We will also use our first MCP client

The MCP Inspector v0.11.0 interface is shown. On the left, there's a sidebar with Transport Type (STDIO), Command (npx), Arguments (@modelcontextprotocol/server-env), and Logging Level (debug). The main area has tabs for Resources, Prompts, Tools, Ping, Sampling, and Roots. The Tools tab is active, displaying a list of tools: echo, add, printEnv, longRunningOperation, and others like List Tools and Clear. The echo tool is selected, showing its description: "Echoes back the input". A message field contains "ping", and a "Run Tool" button is present. Below it, a "Tool Result: Success" section shows the output "Echo: ping". The History tab shows a list of recent operations: 6. tools/call, 5. tools/list, 4. ping, 3. resources/templates/list, and 2. resources/list. The Server Notifications tab shows a list of notifications: 13. notifications/message, 12. notifications/message, 11. notifications/message, 10. notifications/message, and 9. notifications/message.

- Install npm using the following link first
<https://nodejs.org/en/download>
- Install and open the MCP inspector with the following command:
`npx @modelcontextprotocol/inspector`
- For more info check
<https://github.com/modelcontextprotocol/inspector>



Source: <https://github.com/diax12/BUILDING-AND-DEPLOYING-AI-AGENTS---Part-2/tree/main/classes/class-07-mcp-protocol/demos/intro-mcp-walkthrough>

Demo – Adding MCP to LangGraph Agent

The screenshot shows a Streamlit application interface. On the left, there's a sidebar titled "Controls" with a "Reset conversation" button. The main area has a dark background with white text. At the top right, there are "Deploy" and three-dot menu icons. The title "LangGraph MCP Agent Sandbox" is centered. Below it, a sub-instruction says "Ask questions, and the agent will decide when to call MCP tools exposed by `mcp-server.py`". A blue message box contains the text "You are an AI assistant that can call MCP tools to inspect files inside this project." Another message box below it says "Summarize the content of the README.md". A detailed note explains: "The README.md describes a demo that integrates a LangGraph ReAct agent with a Model Context Protocol (MCP) server. Users can interact with the agent through either a command-line interface (CLI) or a Streamlit chat UI." A section titled "Key Sections:" lists three main points: 1. Setup Requirements, 2. Environment Variables, and 3. Usage Options. Under "Usage Options", it details "CLI Check" and "Streamlit Chat UI". At the bottom, there's a text input field "Write the summary into a file" with a red border and a right-pointing arrow icon.



Source: <https://github.com/diax12/BUILDING-AND-DEPLOYING-AI-AGENTS---Part-2/tree/main/classes/class-07-mcp-protocol/demos/langgraph-agent-with-mcp>

Demo – Leveraging the MCP community

Sharing is caring

TOOLHIVE

Integrations Security Features Contact Docs [Download](#)

MCP Servers Made Simple and Secure

One-click deployment with built-in security. No complex setup. No risky config. Just safe and easy server management.

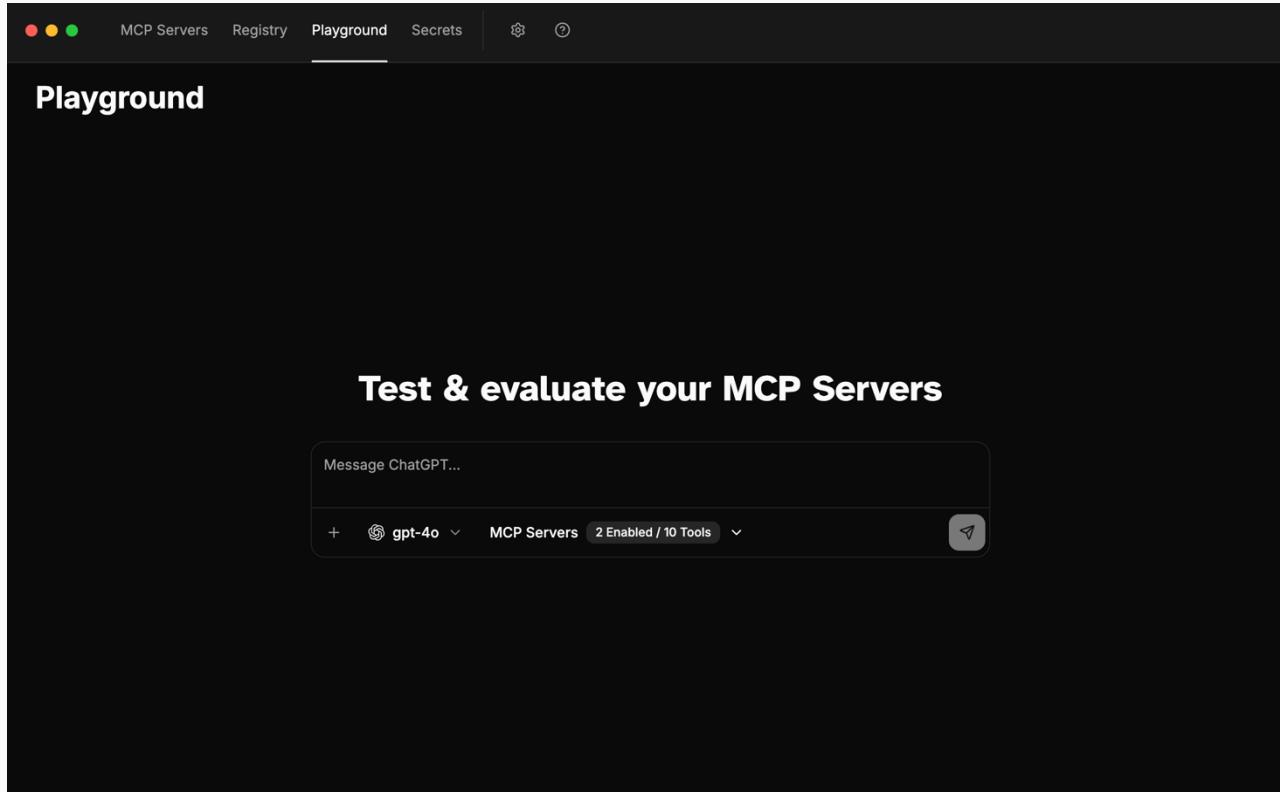
[Download for free](#)
macOS – Arm

Works with VS Code Cursor Copilot Cline Roo Code Claude Code

ToolHive is a great tool to install, configure and use MCP servers with different clients

We can even quickly test the MCP server through a ChatUI

TΛ!



Using the ChatUI is a great way to [assess](#) if the [MCP server](#) will be relevant for the AI Agent we are developing. [Build fast, fail fast!](#)

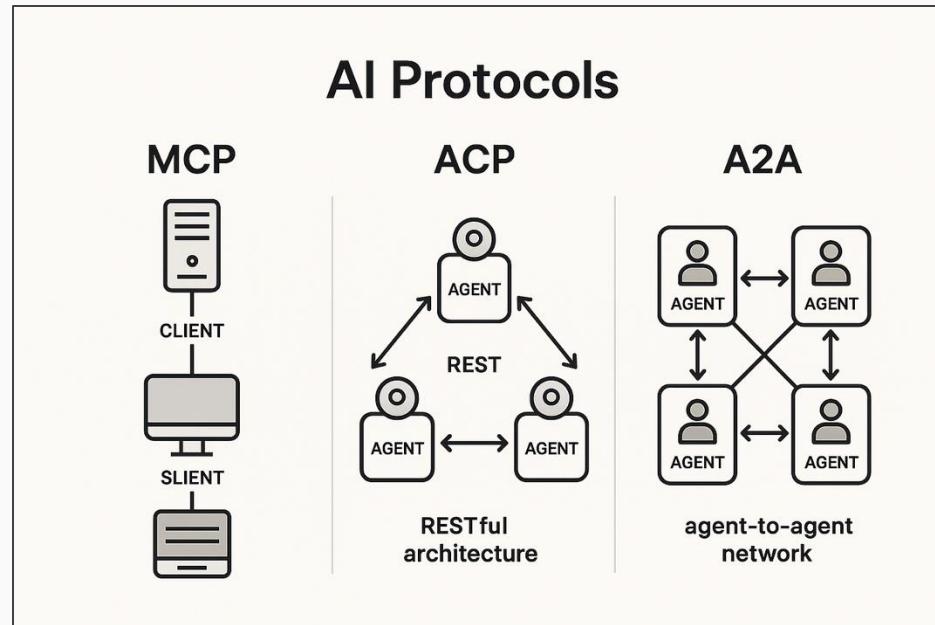
The protocol wars

What other AI protocols already exist?

Three Major Players Competing/Complimenting?

- **MCP (Anthropic)**: AI-to-tool communication and data access
- **A2A (Google)**: Agent-to-agent system communication, secure collaboration
- **ACP (IBM Research)**: Agent Communication Protocol, focuses on practical adoption first

The Stakes: Who becomes the "HTTP" of AI agent communication?



AI Agents in the Real World

Where Agents Are Making an Impact



"We believe that, in 2025, we may see the first AI agents 'join the workforce' and materially change the output of companies."

Industry	AI Agent	Key Outcomes
1. Customer Service	H&M's Virtual Shopping Assistant	40% reduction in cart abandonment, 3x boost in conversion
2. IT Operations	IBM Watson AIOps	60% faster incident resolution, 80% drop in false alerts
3. Cybersecurity	Darktrace Autonomous Response	Real-time threat neutralization, 92% breach reduction
4. Healthcare	Mass General Brigham AI Copilot	60% less documentation time , improved patient engagement
5. Drug Discovery	Insilico Medicine AI Agents	Reduced R&D costs by 70% , accelerated molecule discovery
6. Manufacturing	Siemens Industrial Edge Agents	30% downtime reduction, predictive maintenance
7. Inventory	Ocado Smart Fulfillment Agent	99.9% order accuracy , real-time stock updates
8. Supply Chain	DHL Resilience360 AI Agents	35% fewer delays , improved supplier communication
9. Finance	Bank of America's "Erica"	1B+ interactions, 98% issue resolution rate
10. Government	Singapore's "Ask Jamie"	50% call deflection, 15M+ questions answered , improved citizen satisfaction

You can find more use cases here: <https://tinyurl.com/learning-tutai>



Exploring using cases across your company

Functions					
	R&D	Procurement	Supply chain	Marketing and sales	Customer service
Horizontal use cases Enterprise-wide	Employee copilots Employee use of copilots, such as Microsoft 365 Copilot, Google Workspace AI, and Zoom AI Companion				
	Chatbots Internal deployments of general-purpose tools, such as ChatGPT, Gemini, and Claude				
Vertical use cases Function-specific	Idea generator Summarizes research trends and proposes areas to explore	Negotiation assistant Suggests negotiation levers and next best actions	Supply risk assessor Generates risk profiles for suppliers to boost sourcing resilience	Sales assistant Prioritizes and qualifies accounts	Support assistant Supports staffers with suggested replies and knowledge base lookups
	Research curator Auto-tags and organizes incoming scientific articles by topic	Request for proposal (RFP) generator Auto-generates complete RFPs	Demand forecaster Predicts demand based on internal and external data	Insights provider Provides campaign performance insights based on predefined KPIs and frameworks	Ticket categorizer Prioritizes tickets based on level of urgency
					Generates meeting minutes, writes emails, and translates text Secured access to external GPT capabilities enriched with company-specific data Function-specific copilots, triggered by a human prompt Individual task automation within a business workflow, triggered by a human prompt

Source: <https://www.mckinsey.com/capabilities/quantumblack/our-insights/seizing-the-agentic-ai-advantage>

Luis this is nice but seems to high level!

T^{A'}



I will share my experience with one of TUI's Agent in production

Case Study: TUI Travel Assistant

- **Problem:** Scale personalized travel advise across markets
- **Solution:** Bedrock + Guardrails + Internals APIs + Smartphone App
- **Outcome:** Assisting the customer on booking his next holiday with TUI

Live Demo

I will walk you through a real implementation and then show you a simplified version built using what we learned on this module. Pay attention to the architecture decisions and trade-offs made during development.



TUI Travel Assistant

Real-world application





Global group -
headquartered in Germany

Cruise ships



16

Hotels



~400

Aircraft



~130

Travel Agencies



~1200

Revenue 2024:
23.2 bn €

Underlying EBIT 2024:
1.3 bn €



TUI Care Foundation, initiated by TUI, promotes the positive effects of tourism in 25 countries



140 million API calls per day
5356 TB data traffic per month
10 billion lambda calls per month
Serving 20+ million customers



Over 7 million TUI Collection excursions have been sold since the launch

Motivation

As a TUI App user

I want assistance when trying to find my next holiday

So that I can figure out when and where to go

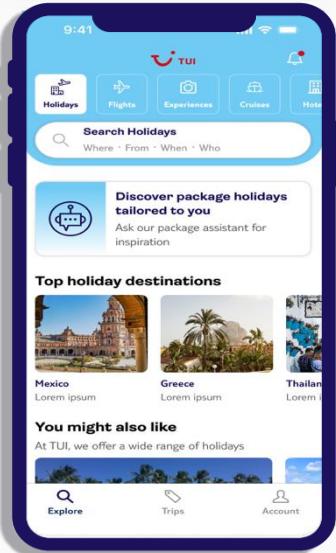
Customer goal: Replicate the retail experience in your pocket. Perfect for those struggling to decide where or when to go, let TUI support you through the difficult decision-making process and find you the perfect holiday.

Business goal: Improve breadth of data and knowledge on customers, enhancing personalisation models. Better understand the issues customers face when finding products to enhance experience in the future.

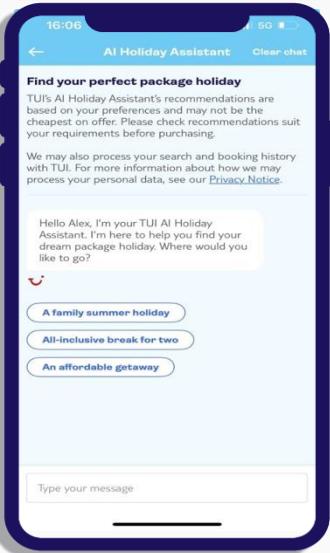


AI Conversational search

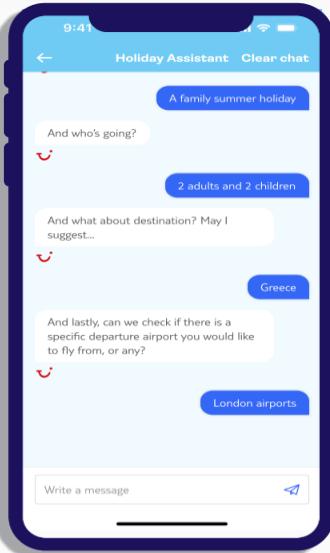
T^A



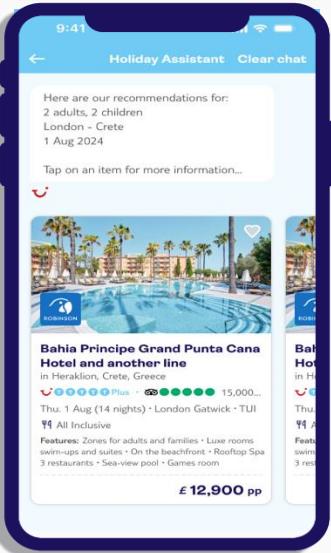
Entry Point A TUI customer opens & logs into the app, ready to plan their next holiday. The customer spots the Holiday Assistant homecard, intrigued, and clicks to navigate into the feature.



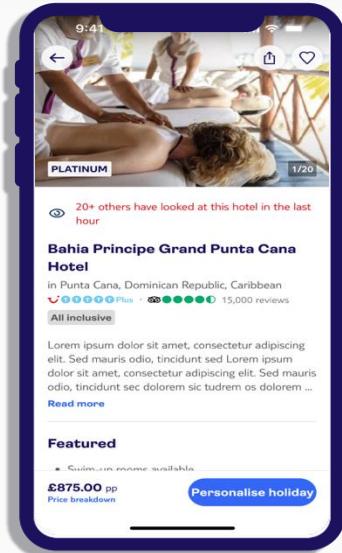
Chat Interface The chat pops up, asking, "Hi Joe, let us help you find your next holiday!" The chat then asks Joe what he's looking for with, with either pre-populated suggestion that he can pick from, or he input text in the field below.



Q&A Joe then clicks on a family summer holiday, and the chat starts to ask Joe for more details, including how many guests, where to depart from and any country or cities he had in mind.



Tailored Suggestions The agent processes this and suggests actual package holidays based on Joe's input. The results are shown in a card carousel which Joe can scroll and compare with the summary shown on the card, including the price.



Holiday Details Joe is then interested in one of the suggested holiday packages, he clicks on the card and lands on the native holiday details page. From there, he can proceed with configuration or go back to the chat.

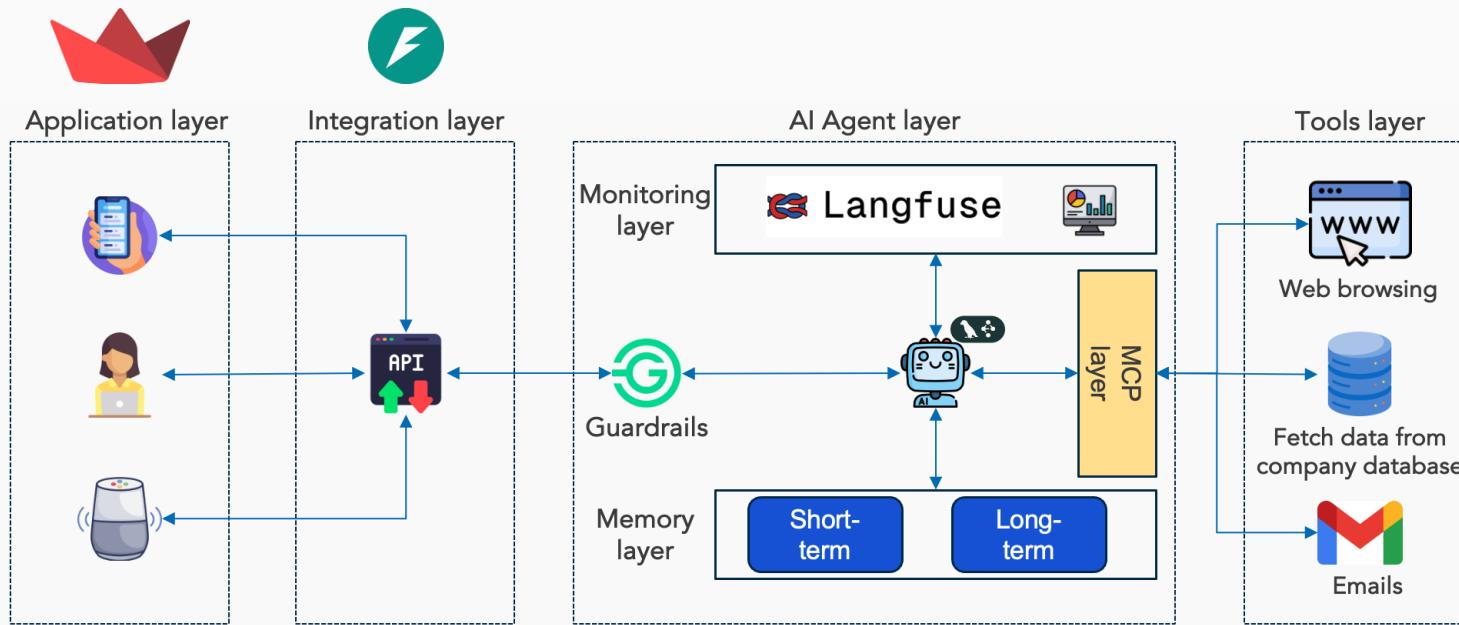
TUTAI Travel Agent Live Demo



Let's connect all the dots and build an AI Agent Application end to end

TΛ!

This demo will leverage all you have learned in this module



Source: <https://github.com/diax12/BUILDING-AND-DEPLOYING-AI-AGENTS---Part-2/tree/main/classes/class-07-mcp-protocol/demos/ai-agent-application>

A travel planner that builds the perfect itinerary

T^Λ

The screenshot shows a travel planning application with a dark theme. On the left, there's a sidebar with sections for 'Account' (Signed in as student, Log out), 'Backend' (FastAPI base URL: http://127.0.0.1:8000, Reset session), and a 'Trip Overview' section. The main area is titled 'MCP Travel Planner' and includes fields for Destination (Munich), Budget (USD) (1500), Number of days (5), Trip start date (2025/12/01), and a text area for Travel style and must-dos (I want to visit the essential landmarks in the city and visit the best Christmas markets according to the locals). A red 'Generate itinerary' button is present. Below this is an 'Itinerary' section with a document icon. The 'Trip Overview' section details the destination as Munich, Germany, the duration as 5 Days (December 1 - December 5, 2025), and the budget as \$1500 USD total. The 'Weather Forecast' section lists December average temperatures (0°C to 5°C / 32°F to 41°F) and conditions (expect cold weather with a chance of snow, pack warm clothing). The entire interface is set against a dark background.

You can access the travel app here: <https://diaz-travel-app.streamlit.app/>

Reflection

- How are you planning to leverage MCP for your capstone project?
- What aspects of the **TUI Travel Assistant** architecture could you reuse in your capstone project?
- How will you define success metrics (KPIs) for your agent? Which ones matter most?
- What were the main architectural lessons learned from the TUI case study (e.g., memory, tools, supervision pattern)?
- How can scalability and latency challenges be mitigated in your own design?

M6 assignment

Focusing on the technical side of the capstone project

1. Build your **LangGraph-based AI Agent** using the class examples as a starting point.
2. Integrate **at least one MCP server** and **one additional tool** of your choice.
3. **Deploy** the agent API on **Render** and connect it to a **Streamlit Community Cloud UI**.
4. Add **Guardrails AI** to enforce safety (e.g., input validation or output moderation).
5. Set up **Langfuse** for monitoring and evaluation (track latency, tool calls, quality).
6. Reflect on design trade-offs and summarize lessons learned.
7. Submit by email including:
 - GitHub repo (source code + README)
 - Render API & Streamlit app links
 - Langfuse dashboard or screenshots
 - PDF report (\leq 10 pages: architecture, deployment, evaluation, guardrails, reflection)



Deadline: December 20, 2025 (aligned with Capstone Delivery Date)

Wrap-up

Developing an AI Agent goes beyond building it

- Your AI Agent should deliver **reliable, measurable**, and **safe outputs**.
- It must be **observable**, and **aligned with real user needs**.
- From day one, you should be able to **track its success**.

Next:

- Begin defining your **Capstone use case** and the real-world problem your Agent will address.
- Start planning your **technical implementation** for M6, which will focus on evaluating the Agent you build.
- (Optional) Assign **one Technical Lead per team**. This person will coordinate the technical work (LangGraph, MCP, Render, Streamlit, Langfuse, Guardrails AI).
- **Remember** that your Capstone story and your M6 Agent evaluation are **two sides of the same project**.

In short: Your capstone project defines the “why” and M6 demonstrates the how





Questions?



luis.f.s.m.dias@gmail.com



<https://www.linkedin.com/in/luisfilipedias/>