One-Stop Medical Hub

Overview

This project implements a MCP (Model Context Protocol) server that connects Claude (an LLM application) to the medical drone delivery REST service built in the previous coursework. Through this integration, users can interact with the delivery system conversationally to access medicine requirements, stock availability, delivery options, and even visualise potential flight paths. They can also place orders through a single prompt, with a lightweight persistence layer demonstrating how Claude can streamline order recording. Overall, the MCP server acts as an interface layer, translating natural-language prompts into appropriate API calls.

Problem Statement

The existing REST service exposes its functionalities through REST API endpoints so users who are not familiar with backend tools or API calls cannot interact with the service easily, limiting its accessibility and usefulness real-world contexts.

Who does it benefit?

As of now, this project benefits any non-technical stakeholders, such as potential customers, medical staff, or administrators, who may wish to give feedback on the REST service before it is deployed.

In the long term, it serves as a prototype for a future user interface by demonstrating how conversational interaction can replace manual forms or complex UI flows. As shown in the current version, users can already place orders seamlessly through plain language, and with further enhancements, administrators could also update medicine requirements, adjust stock levels, or modify pricing without editing configuration files or using administrative dashboards. This streamlines both customer and admin workflows into a single, intelligent, natural-language-driven interface.

Why this solution?

The inspiration for this project stems from the growing popularity of AI assistants over traditional search engines. People increasingly prefer instant, summarised, and conversational responses. This project does exactly that, making the system far more accessible to non-technical users and allowing them to explore features as if speaking to a human.

Implementation

The core functionality of the MCP server comes from MCP tools, which call the relevant REST API endpoints to return the requested information or perform the action specified by the user's prompt. I used the MCP Python SDK [1], as it simplifies the creation of such tools.

For e.g. the drones_with_cooling tool allows the LLM to answer a prompt such as *"Which drones have cooling capability?"*. A clear docstring specifying the tool's purpose, arguments, and return data enables the LLLM to call it automatically with the parameter state set to true. Internally, the tool sends a GET request to /api/v1/dronesWithCooling/{state} and returns the backend response in a user-friendly format.

```python
@mcp.tool()
async def drones_with_cooling(state: bool) -> dict:
    """
    Get a list of drone IDs filtered by cooling capability.
    Calls: GET /api/v1/dronesWithCooling/{state}

    Args:
        state: True for drones WITH cooling, False for drones WITHOUT cooling

    Returns:
        A list of drone IDs
    """
    url = f"{BASE_URL}/dronesWithCooling/{state}"

    try:
        async with httpx.AsyncClient() as client:
            response = await client.get(url, timeout=10)
            response.raise_for_status()
            drones = response.json()
            return{
                "drones": drones,
                "count": len(drones)
            }
    except Exception as e:
        return {
            "count": 0,
            "drones": [],
            "error": str(e)
        }
```

Additional MCP tools

- **place_orders**

This tool is accessed when the user wants to place an order.
Prompt: "*Place an order 123, delivery time 14:30 on 22/12/2025, capacity 0.5, max cost 50, going to lat 55.94440 and lng -3.19031."*

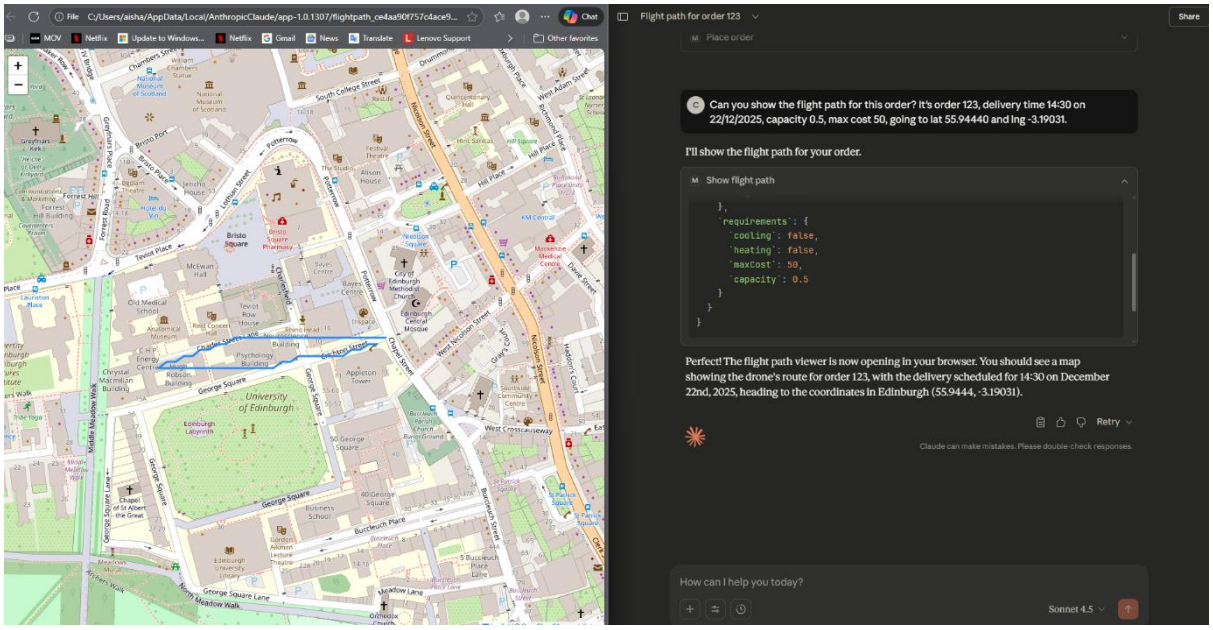
I added a persistence layer using SQLite because it is lightweight and sufficient to show how Claude behaves for this prompt. This involved adding the JPA starter and the JDBC driver dependencies to the pom.xml, specifying the database file path in application.yml, and creating an entity to map incoming orders to the database, along with a corresponding repository for storage, retrieval, and search operations [2]. A new POST endpoint, /placeOrder, was implemented. It checks which drones are available for the requested delivery and, if a drone is not already assigned to another order, it saves the new order and returns a confirmation message. If all drones are currently in use, Claude informs the user to try again later.

The drone-assignment logic is intentionally simple, serving only to demonstrate how an order request made through natural language is processed and stored behind the scenes.

- **show_flight_path**

Claude accesses this tool to help a user visualise a potential delivery route.
Prompt: "Can you show the flight path for this order? It's order 123, delivery time 14:30 on 22/12/2025, capacity 0.5, max cost 50, going to lat 55.94440 and lng -3.19031?"

The calcDeliveryPathAsGeoJson endpoint from the previous coursework returns the proposed path as GeoJSON. The MCP server uses the Folium Python library as it supports quick conversion of GeoJSON objects into an interactive Leaflet.js map [3]. The map is saved as an HTML file by the server, and Claude automatically opens it in the user's browser for visualisation.



- **get_medicine_info**

This tool allows Claude to provide users with details about a medicine in stock, such as capacity needs or storage conditions.
Prompt: "What are the medicine requirements for insulin?"

For this feature, I added a small JSON file that stores some medicines and their associated requirements. When the REST service starts, this file is loaded into memory by the service/MedStockService, and the data is mapped into MedInStock DTOs. A new endpoint, /medicineRequirement/{name}, exposes this information, and the MCP tool simply calls it.

- **drones_available_for_medicine:**

It is used by Claude to inform the user which drones can complete a particular order.
Prompt: "I need to check drone availability for a medicine order with ID 1, delivery date 2025-01-02 at 12:00, requires capacity 4.0, and must be delivered to (55.941, -3.280)."

The old queryAvailableDrones endpoint is called to return a list of available drones that satisfy the order's requirements and do the delivery on time.

- **delivery_possible_at_location:**

A user may want to know whether the delivery service is accessible in their region.
Prompt: "Is delivery possible at 55.94523, –3.18730 or do I not have access to this medical drone delivery system."

The ray-casting algorithm from coursework 1 is used to check whether the location in the prompt lies within a no-fly region.

Other Considerations

The stdio communication protocol was used to allow Claude to communicate with the MCP tool server, so no network setup was needed. This also allowed the REST service to be run locally for the proof of concept.

Use of Generative AI

ChatGPT [4] was used to improve docstrings, generate example prompts, and assist with loading JSON medicine data into DTOs. Using an LLM for docstrings was especially helpful because MCP tools rely on clear instructions that another LLM (Claude) must interpret correctly. So ChatGPT's responses helped me better understand how tool descriptions would be used during execution.

996 words

References

[1] Anthropic, "python-sdk," Anthropic, 2024. [Online]. Available: https://github.com/modelcontextprotocol/python-sdk?tab=readme-ov-file. [Accessed 27 November 2025].

[2] A. Obregon, "Using Spring Boot with SQLite for Lightweight Apps," Medium, 8 June 2025. [Online]. Available: https://medium.com/@AlexanderObregon/using-spring-boot-with-sqlite-for-lightweight-apps-6c7624a0f438. [Accessed 27 November 2025].

[3] R. Story, "Using GeoJson," GitHub, 2013. [Online]. Available: https://python-visualization.github.io/folium/latest/user_guide/geojson/geojson.html. [Accessed 27 November 2025].

[4] OpenAI, "ChatGPT," OpenAI, 2025. [Online]. Available: https://chat.openai.com/. [Accessed 27 November 2025].