OpsReady MCP AI Agent

Project Overview

Originally developed as a team project for OpsReady in December 2025. After the course ended, I independently converted the prototype into a production-ready full-stack application.

Original Team

- Aryan Joglekar
- Blake Alberding-Berge
- Gesi Morris-Odubo

Phase 1: Team Prototype (Original Prototype)

1. Problem Description:

1.1 Company:

The company we are working with is OpsReady. OpsReady is a SaaS company that provides organizations with a centralized all in one system for managing their operational data. Companies using OpsReady are given their own site, which they can access on any browser or the OpsReady mobile app. Inside of a company's site, they can organize their data in workspaces, which primarily are location based, such as a jobsite, but workspaces can be created for anything. Workspaces are what stores operational data such as forms, incidents, asset lists, inspections, tasks, and employee activity. OpsReady sites also allow managers and supervisors to assign tasks to employees, allowing for efficient operational management. Employees can complete tasks, inspections, and incident reports and have all of the data be stored in the workspace, along with automation such as task creations if a damaged asset is reported. OpsReady is designed as a solution to allow companies one platform to manage all of their data and operations.

1.2 People:

Inside of OpsReady we are working with a few key people:

1) Michael Smith:
    a) Michael is responsible for AWS and OpsReady's infrastructure. Michael has been our main point of contact in meetings and has assisted us with getting access to AWS, demo site, API documentation, and many more things.
2) Stanley Liang:
    a) Stanley is the Senior Product Manager at OpsReady. Stanley has also been a main point of contact during weekly meetings, answering broad questions and guiding us on the project scope, direction and ideas.
3) Jennifer Mackie:

a) Jennifer is a Senior Software Developer, helping us with technical questions and guiding us through understanding the API documentation and code questions.

1.3 Problem:

With companies using OpsReady's services generating so much data, it makes it extremely hard for them to understand all of their data. Since all of the data generated is directly related to operations, there is a ton of data that can benefit the company. It is difficult for companies to understand their data and see how different data points relate to each other. Companies require a simple, efficient way to query and analyze their data, without manually having to look through large datasets. We were tasked with developing a proof of concept AI agent that managers can interact with to retrieve and answer questions about their data and daily operations.

2. Requirements:

2.1 Functional:

The functional requirement that we gathered through the first few weekly meetings with the client were:

- The agent must understand conversation-style user input, and be able to respond to users naturally
- Agent must respect user permissions / authenticate each query, to ensure users can access the site.
- Agent should not expose data to LLM for testing
- Agent should be able to handle complex queries by linking API calls together
- Agent should be able to process form data

2.2 Non Functional

The non functional requirement that we gathered through the first few weekly meetings with the client were:

- Data-retrieval should not require full database access, should use the existing APIs
- Easy to user and attractive user interface

3. Analysis

After understanding the problem and the requirements, we had 2 main approaches that we tried to solve the problem.

Approach 1 - AWS Bedrock:

- Using AWS bedrock, we would choose a model that we want to use as our LLM, train it on the company data and use this to answer queries. This proved to be difficult without a database for the model to be trained off of. Without database access we were limited with the amount of data needed for training, which requires a lot.
- Bedrock is for cloud based models, which would be great so all of the team could share the same model, but it raised questions about testing and how many tokens would need to be generated for testing.
- Our team also had limited exposure to Bedrock and we were concerned we would spend a majority of the time understanding how to implement it.

Approach 2 - Model Context Protocol and Claude (MCP):

While talking with the client during a weekly meeting, we were discussing MCP and how it could benefit our project. MCP allows for tools to be created, each tool represents a different function that the agent can perform by calling an API. We also discussed using a pre-trained LLM to act as the front end, which the client approved as it would take too long to train our own LLM to speak naturally. We decided to use Claude as our model of choice, as it integrates seamlessly with the MCP approach and has an attractive UI. This approach fit our project great, and solved a key issue we had with not providing database access. Using MCP there was a few components we needed to design:

MCP Server - Python:

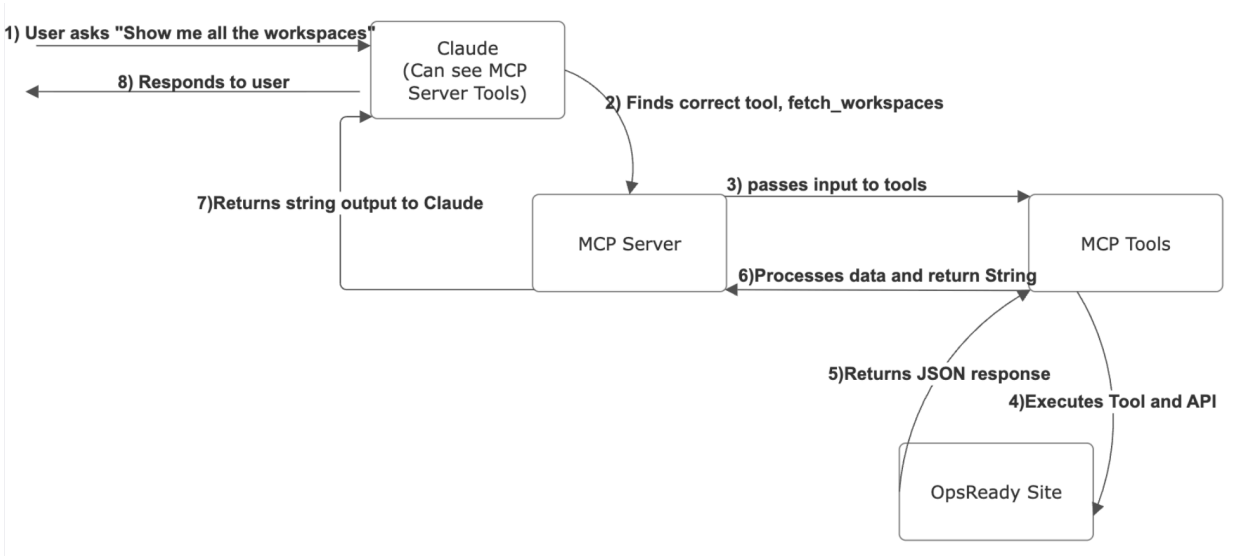
- The Server would act as the bridge between our backend logic and Claude, only exposing tool declarations to Claude. Meaning Claude can not see the Authentication or API calls, which is a great security feature
- The server contains all of our tool declarations, which are JSON formatted input schemas that describe what the tool does, parameters, and the name.
- The Server also is responsible for calling the selected tool function, which then calls the API, extracts the appropriate data and returns it back to the Server.

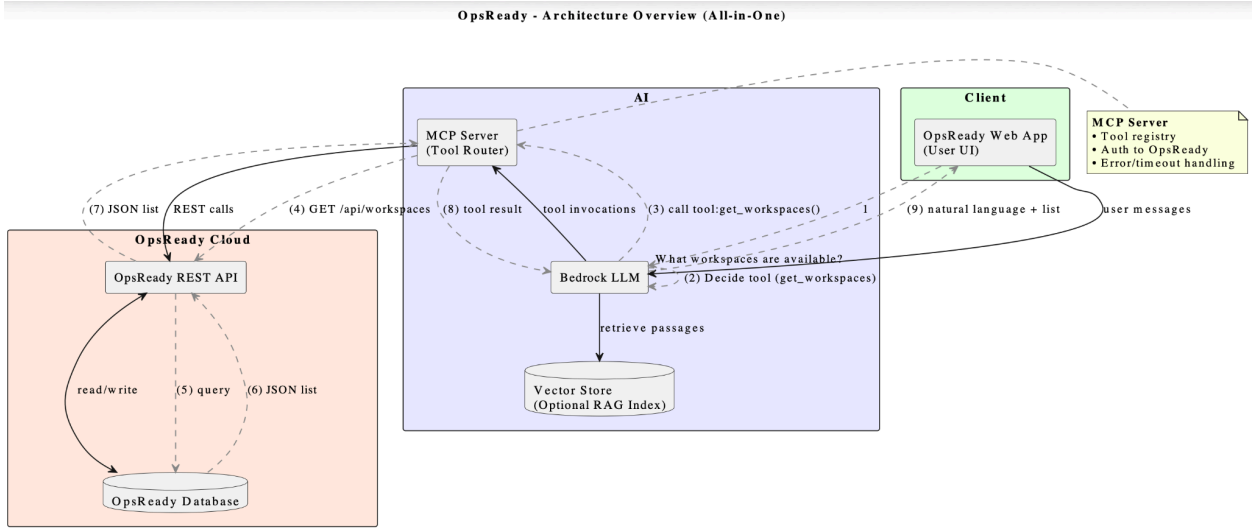
MCP Tools - Python:

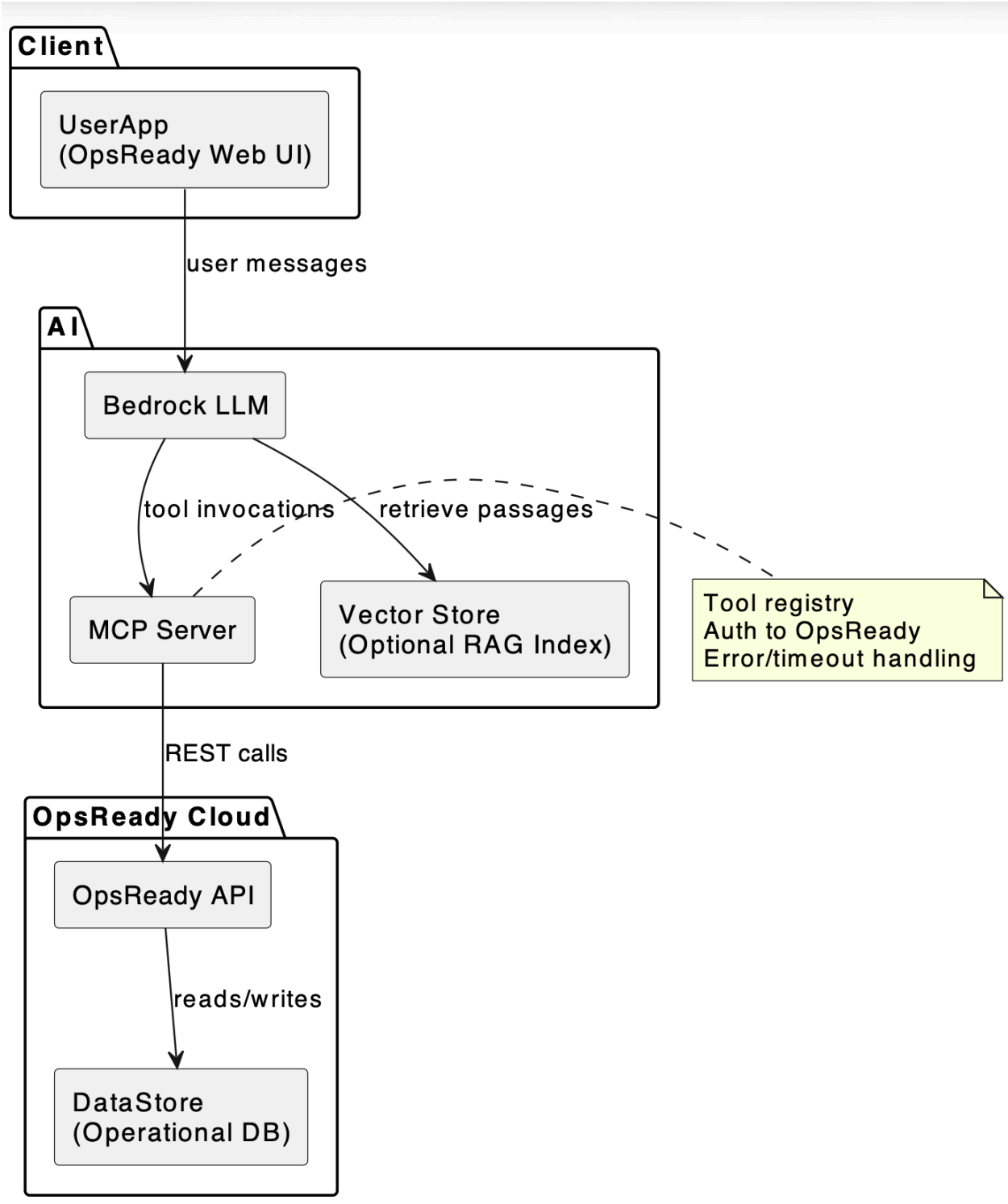
- Tools act as tasks that the agent can perform, they call the API and return the data back to the server, where Claude then receives the data and outputs it to the user
- Tools can be easily added to the model, allowing for scalability.

MCP and Claude was the approach we decided to use.

4. Design






Client
UserApp
(OpsReady Web UI)
user messages
AI
Bedrock LLM
tool invocations
retrieve passages
MCP Server
Vector Store
(Optional RAG Index)
Tool registry
Auth to OpsReady
Error/timeout handling
REST calls
OpsReady Cloud
OpsReady API
reads/writes
DataStore
(Operational DB)

5. Implementation

5.1 Read Me

OPSREADY MCP AGENT

Welcome to the OpsReady MCP Agent, a proof-of-concept AI assistant designed to help organizations effortlessly understand and interact with their operational data. This project blends modern AI, backend engineering, and the **Model Context Protocol (MCP)** to create a seamless, natural-language interface for querying and analyzing real-world operational datasets.

Table of Contents

Introduction

This project aims to make it simple and intuitive for companies using OpsReady to query and understand the operational data they generate. It demonstrates how an AI-powered agent, integrated through the MCP, can retrieve, interpret, and analyze large datasets through natural language—removing the need for manual data inspection.

Features

- **Speech-to-Text Input**
  Hold the **LEFT SHIFT** key to record your voice. The recording is automatically transcribed and saved for the AI agent to read.

- **OpsReady Tools Integration**

The AI agent can interact with various OpsReady endpoints to retrieve operational data:

Tool Function	Description
Get Activity Feed	Fetch recent activity within the workspace.
Get User Logins	List users who have logged in since a specific date.
Get Tasks	Retrieve tasks assigned to users (overdue, upcoming, all).
List Forms	Fetch available forms and their IDs in the workspace.
Get Assets	Retrieve assets associated with a workspace.
Team-Specific Tasks	List tasks assigned to specific teams.
Deficiencies	Retrieve reported deficiencies for a workspace.
Work Orders	Retrieve open or closed work orders.

Technologies Used

Category	Component	Description
Backend & Core	Python 3.10+, fastmcp, requests, dotenv	Main backend language, MCP framework, API communication, credential management.
AI Integration	Model Context Protocol (MCP), Claude Desktop App	Connects your local tools to the AI interface.
Voice Input	Whisper.cpp, pynput, sounddevice, scipy, numpy	Local speech-to-text engine, key detection, audio capture, and file processing.

Design & Architecture

There are two main components to the MCP System we designed. There is a Server.py file that connects our backend tools to Claude, and then there is a Tools folder which contains all of our tools.

Server.py
Server.py is exposed to Claude through a path saved in Claudes config file (more on this in Installation Section). The servers role is to define all of our tools in JSON Schema formatting. There is a python function called list_tools() that contains all of the tools that Claude can access. @app.list_tool() as seen in the first line of the code sample below, is MCP's way of telling the model, in our case Claude, what the available tools are.

Example tool declaration that requires a since date, then returns the list of users signed in since that date.

```python
@app.list_tools()
```

```
async def list_tools() -> list[Tool]:
    return [
        Tool(
            name="get_recent_logins",
            description="Returns users who have logged in since
certain date (YYYY-MM-DD)",
            inputSchema={
                "type": "object",
                "properties": {
                    "since_date": {
                        "type": "string",
                        "description": "Date in YYYY-MM-DD format",
                    }
                },
                "required": ["since_date"]
            }
        ),

```

Server.py also contains a python function to call the tools called call_tools(). The function tells Claude to call this function when it wants to execute a tool by including @app.call_tool(), then Claude passes the tool name it wants to execute and any parameters like workspace name to the function.

Inside the call_tool() function there are elif statements that execute the correct too based on the tool name that Claude wants to execute.

Example of the call_tools() function:

```python
@app.call_tool()
async def call_tool(name: str, arguments: dict) -> list[TextContent]:
    """Handle tool calls."""

    if name == "get_recent_logins":
        return await get_recent_logins(arguments["since_date"])
    elif name == "get_user_tasks":
        return await get_user_tasks(arguments["identifier"])
    elif name == "get_task_sample":
        return await get_task_sample(arguments.get("limit", 5))
    elif name == "get_all_assigned_users":
```

```
        return await get_all_assigned_users()
    elif name == "get_overdue_tasks":
        return await get_overdue_tasks()
    elif name == "get_task_summary_report":
        return await get_task_summary_report()
```

So whatever tool name Claude passes as a parameter to be executed, the call_tool() function executes it and then the output is returned back to the Server, then to Claude.

And finally the Server.py contains a main() function that implements MCP helper function stdio_server(), which creates the server as a read_stream and write_stream so Claude can see our server and is able to both read data and write date. The main() function also prints "Server Running" so the user can know its running.

```python
async def main():
    from mcp.server.stdio import stdio_server  #import helper function
from mcp package
    print("Server running")
    async with stdio_server() as (read_stream, write_stream):
        await app.run(read_stream, write_stream,
app.create_initialization_options())   #start the server as read and
write
```

Tools

Tools in MCP can be thought of as "Tasks" where each tool is a different task the agent can perform. Most tools call atleast one API to get the data from the backend site. Tools can call as many APIs as needed, and can even call other tools to get access to their data.

All of the tools that we created are contained in the "tools" folder, which is just a way to keep clean code structure so the files do not get messy. Tools can be kept anywhere.
Each tool needs to be declared in Server.py as in JSON Schema format under the @app.list_tools() section, this is how Claude is able to see the tool, also each tool must be called in the call_tools() function to be able to be executed.

Here is an example of how a tool is setup:

```python
async def get_recent_logins(since_date: str) -> list[TextContent]:
```

```
    try:
        since = datetime.strptime(since_date,
"%Y-%m-%d").replace(tzinfo=timezone.utc)
    except ValueError:
        return [TextContent(
            type="text",
            text="Invalid date format. Use YYYY-MM-DD."
        )]

    url =
f"{BASE_URL}/api/account?limit=500&offset=0&search=&with_teams=true"

    try:
        tgt = get_tgt(USERNAME, PASSWORD)
        if not tgt:
            print("failed to get TGT")
            return [TextContent(type="text", text="Failed to get
TGT")]

        service_url = f"{BASE_URL}/api/login"
        st = get_st(tgt, service_url)
        session = get_api_session(st)
        response = session.get(url)
        response.raise_for_status()
        data = response.json()
    except Exception as e:
        return [TextContent(
            type="text",
            text=f"Failed to fetch accounts: {str(e)}"
        )]
```

This is the tool responsible for returning a list of users that have signed in from a passed date. It takes in the "since" date as a parameter and will return a TextContext List, which is the MCP object that is used to give data to Claude.

Each tool is also required to authenticate when its called, to ensure the user is able to see the data:

```python
try:
```

```
        tgt = get_tgt(USERNAME, PASSWORD)
        if not tgt:
            print("failed to get TGT")
            return [TextContent(type="text", text="Failed to get
TGT")]

        service_url = f"{BASE_URL}/api/login"
        st = get_st(tgt, service_url)
        session = get_api_session(st)
        response = session.get(url)
        response.raise_for_status()
        data = response.json()
```

This code calls functions that are created in our opsready.py, which is responsibe for getting tgt, st, and session instances for the user requesting data.

Prerequisites

1. **Internet Access**
2. **Python 3.10 or higher**: [Download Link](https://www.python.org/downloads/)
3. **Claude Desktop App**: [Download Link](https://claude.com/download)
4. **OpsReady Credentials**: A valid username and password for the OpsReady sandbox environment.
5. **System Tools** (Required for Whisper compilation): **CMake** and **FFmpeg**.

Installation and Setup

1. Prepare the Environment & Install All Libraries

1. **Download** the repository and navigate to the project folder.
2. Run this single command to install all Python libraries needed for the core server and voice listener:

    ```bash
    pip install mcp python-dotenv requests typing sounddevice scipy numpy pynput
    ```

2. Voice Input Engine Setup (Whisper.cpp)

The voice feature requires system dependencies and compilation.

2A. Install System Dependencies (CMake & FFmpeg)

Operating System	Action	Command/Instructions
macOS	Use Homebrew to install packages:	`brew install cmake ffmpeg`
Linux (Debian/Ubuntu)	Use the `apt` package manager:	`sudo apt update && sudo apt install cmake ffmpeg`
Windows	**Manual Installation and PATH Configuration:**	**[See Detailed Windows Steps Below]**

Windows Detailed Installation Steps

1. **CMake:** Download and install the Windows x64 Installer from [cmake.org]. **CRUCIAL:** Select **"Add CMake to system PATH for all users"** during installation.
2. **FFmpeg:** Download a recent Windows build (e.g., from Gyan). Unzip the folder to a stable location (e.g., `C:\ffmpeg`).
3. **Configure PATH:** You must add the path to the FFmpeg `/bin` directory (e.g., `C:\ffmpeg\bin`) to your Windows **system environment variables** (under **Path** variable).

2B. Compile Whisper

Run these commands from the root of your project directory.

1. Download the Whisper source code:
   ```bash
   git submodule update --init --recursive
   ```
2. Compile the executable:
   ```bash
   cd background/whisper.cpp
   mkdir build && cd build
   cmake ..
   make
   ```
3. Download the language model:
   ```bash
   cd ../models
```

```
    curl -O
[https://huggingface.co/ggerganov/whisper.cpp/resolve/main/ggml-base.en.bin](https://huggingface.co/ggerganov/whisper.cpp/resolve/main/ggml-base.en.bin)
  ```
4. **Verify Paths:** The Python tools expect these files to be present: `background/whisper.cpp/build/bin/whisper-cli` and `background/whisper.cpp/models/ggml-base.en.bin`.

3. Authentication & Configuration

1. **Authentication:** In the project root, create a file named **`.env`** and add your credentials:
    ```env
    OPSREADY_USERNAME=your_email@example.com
    OPSREADY_PASSWORD=your_password
    ```

2. **MCP Config File:** Determine the **exact, absolute path** to your project folder. Create a file named **`mcp_config.json`** and paste this, **updating the `cwd` path**:
    ```json
    {
      "command": "python3",
      "args": [
        "server.py"
      ],
      "cwd": "PATH/TO/YOUR/PROJECT/FOLDER"
    }
    ```
   CRUCIAL Make sure to update the `cwd` key with the absolute path to your folder.

3. **Load Config in Claude:**
   * Open Claude Desktop $\rightarrow$ **Settings** $\rightarrow$ **Developer**.
   * Find **"Load MCP Servers"** and click the button to select the **`mcp_config.json`** file.
   * **Restart Claude Desktop** and check the Developer tab to verify the server is "running."

Steps to Run:
- Open a terminal window in the path where server.py is located
- Run command "python server.py" you should see "Server Running"
- Open Claude desktop and ask "What tools do you have available" you will see a list of tools
- Start asking questions to Claude

Usage

To utilize the tools, you must run two separate terminal sessions: one for the core server and one for the voice listener.

Component	Command	Location
1. Core MCP Server	`python3 server.py`	Project Root Directory
2. Voice Listener	`python3 background/voice_output.py`	Project Root Directory

Command Input

Method	Action	Example
Voice Command	Hold **LEFT SHIFT** to speak. Release to transcribe. Then, type `"Get voice input"` into the Claude chat box.	*Query:* "List all assets." *Claude Input:* `Get voice input`
Text Input	Directly type your query into the Claude chat box.	`Open workorders for Summit Base`

> **Stop Servers:** Stop both running servers using **CTRL + C** in their respective terminal windows.

Adding Tools
The way we designed our MCP server makes it very easy to add and remove tools from Claude.

Adding to Servery.py

Usually the first step to adding a new tool is to add the tools description to Server.py. Although there is no required order to adding a new tool, adding the tool description first allows you to understand what the tool needs.

Use this template to add a new tool to Server.py. The tool description must be added in the function list_tools() function, as described in [Design & Architecture](#design--architecture).

```python
        Tool(
            name="tool_name_here",
            description="Tool description here (what the tool does)",
            inputSchema={
```

```
                "type": "object",
                "properties": {   #Paramaters your tool will take
                    "param_name (any name) ": {
                        "type": "string",    #type of paramater (Int,
String, etc)
                        "description": "Description of the paramater
here",
                    }
                },
                "required": ["since_date"]     #any paramaters that
are required must be set as required
            }
        ),
```

Additionally, in Server.py you need to add the actual tool call

Use this template to add a tool call. The call must be added in the function call_tool(), as described in [Design & Architecture](#design--architecture).

```python
@app.call_tool()
async def call_tool(name: str, arguments: dict) -> list[TextContent]:
    """Handle tool calls."""
    if name == "get_recent_logins":
        return await get_recent_logins(arguments["since_date"])
```

Adding to Tools Folder

As of now, all of our tools are kept in the folder "tools", this is just for code cleanliness and does not need to be kept this way.
To add a tool there a few things that must be done:

Create a new python file for your tool (ex: tool_get_users.py)

Import the required packages and methods

```python
import os
from typing import List
from dotenv import load_dotenv
from mcp.types import TextContent
```

```
from opsready import get_tgt, get_st, get_api_session   #this allows
us to use the methods we created to authenticate
```
Directly below load the username and password from .env
```python
load_dotenv()
BASE_URL = "https://or-student-sandbox.opsready.com"
USERNAME = os.getenv("OPSREADY_USERNAME")
PASSWORD = os.getenv("OPSREADY_PASSWORD")
```
Then you can create your tool method, ** Use async methods so Claude is not blocked **

Authentication is also needed in each tool, make this your first step of any new tool
```python
async def get_deficiency_details(deficiency_id: str) ->
List[TextContent]:   #takes in deficiency id as string, returns List
of TextContent (needed for MCP output)



    try:
        tgt = get_tgt(USERNAME, PASSWORD)   #call get_tgt() from
opsready file
        if not tgt:
            print("failed to get TGT")
            return [TextContent(type="text", text="Failed to get
TGT")]

        service_url = f"{BASE_URL}/api/login"
        st = get_st(tgt, service_url)     #call get_st() and pass it
tgt and the service url (login url)
        session = get_api_session(st)    #get the authenticated
session

        response = session.get(all_ws_url)
        response.raise_for_status()
        data = response.json()
```
We can see that the method takes in a string which is the deciciency_id, the method is required to take in the paramaters you declared it to accept in Server.py

From here, you can create the tool to call different APIs and use the data as you need to return the appropriate output

License

Data Usage and Retrieval

The data accessed, retrieved, or displayed by this software is not governed by any implied license.

All operational data, credentials, and configuration accessed through the OpsReady APIs remain the exclusive intellectual property of **OpsReady** and the client organization.

* **Data Rights:** Access to the data is governed solely by the terms of service and licensing agreement established between the user's organization and OpsReady.
* **Sandbox Environment:** This tool is designed to retrieve data from the OpsReady sandbox environment (`https://or-student-sandbox.opsready.com`). Use of this tool is subject to the conditions and restrictions governing that specific sandbox instance.
* **Security:** Users are responsible for maintaining the security and confidentiality of their OpsReady credentials stored in the local `.env` file.

6. Testing

6.1 Testing Approach

Testing for the OpsReady MCP Agent focused on validating correct data retrieval, authentication reliability, tool correctness, and integration with the Claude Desktop application.

Testing was performed manually using real OpsReady sandbox data by issuing natural-language queries through Claude and verifying that tool outputs matched expected API responses and, where applicable, the OpsReady web interface.

6.2 Authentication Testing

What we tested:

- CAS authentication flow (TGT → ST → session cookie)
- Session reuse across multiple tool calls
- Handling of authentication failures (401 errors)

How:

- Manually triggered tools multiple times in sequence
- Forced invalid credentials to verify failure handling
- Tested re-authentication on expired sessions

Authentication testing verified the correct execution of the CAS login sequence, including retrieval of the Ticket Granting Ticket (TGT), Service Ticket (ST), and authenticated session cookie.

Tools were tested sequentially to ensure sessions remained valid and refreshed correctly when required.
Error handling was validated by simulating failed authentication scenarios.

6.3 Tool Level Functional testing (examples of some tools and how they were validated)

Tool	Test Performed	Expected Result	Outcome
Recent Logins	Query users since a date	List of users with last_login ≥ date	Pass
User Tasks	Query tasks by username	Correct task list returned	Pass
Overdue Tasks	Compare due dates vs today	Only overdue tasks returned	Pass
Unassigned Tasks	Identify tasks without assignee	List of unassigned tasks	Pass
Users Without Tasks	Cross-check users vs tasks	Users with zero tasks	Pass
Task Summary Report	Aggregate all tasks	Correct counts returned	Pass

Data Validation Against OpsReady UI

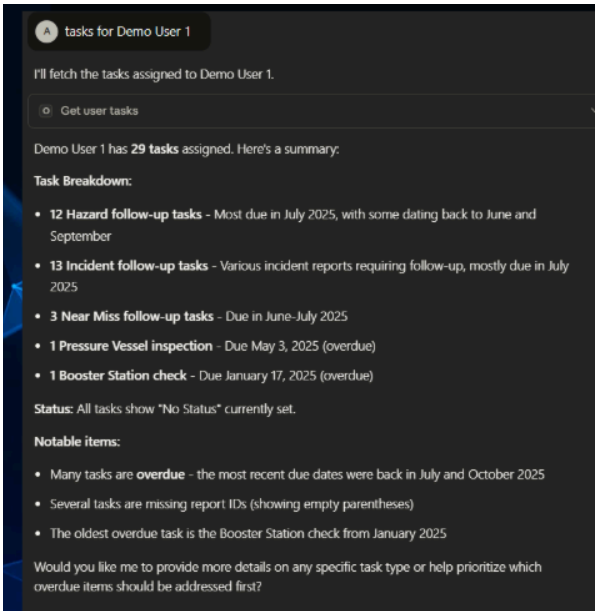
What we tested:

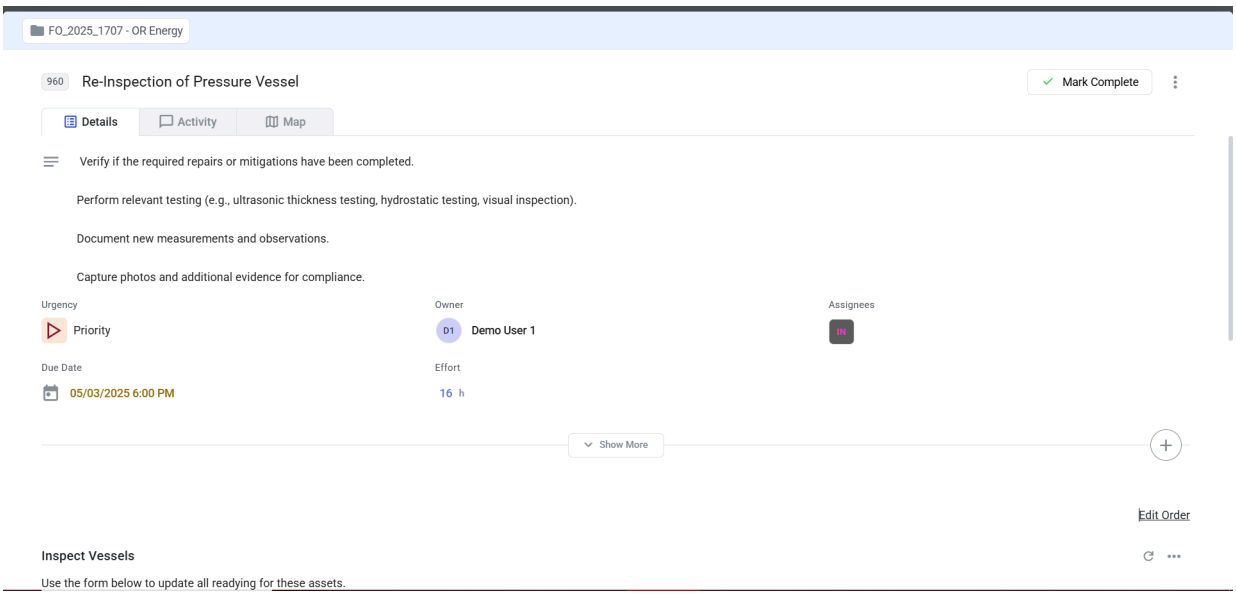
- Compared tool results with the OpsReady website

- Checked task counts, user lists, and overdue tasks

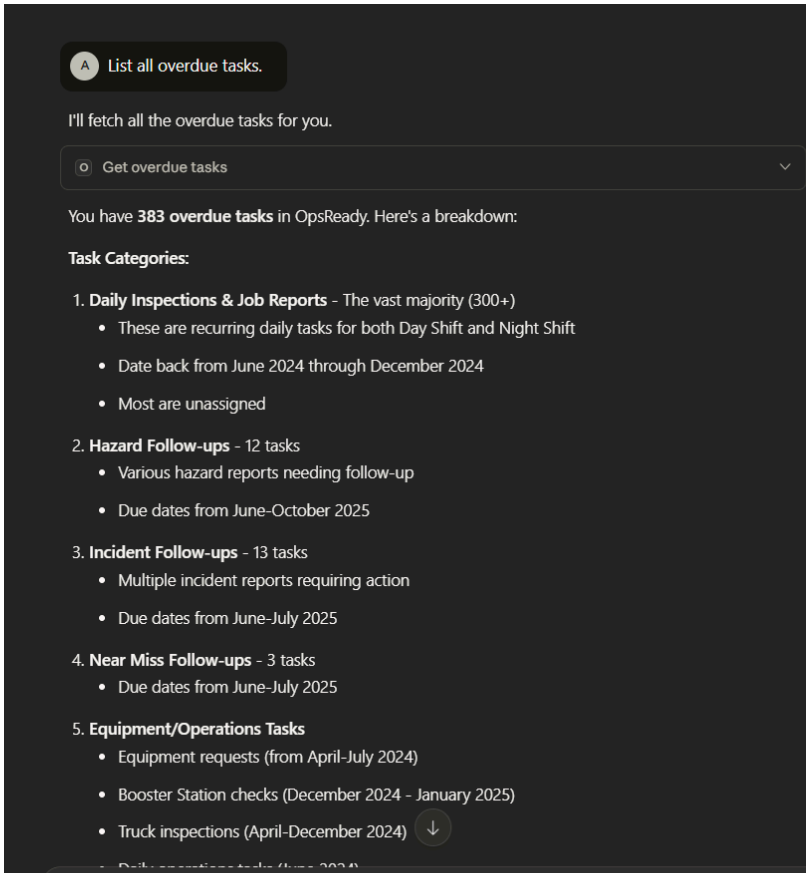
Tool outputs were compared with the OpsReady web interface where possible.
In some cases, API results did not directly match the UI due to system-generated tasks, archived workspaces, or UI-level filtering.
This behavior was expected and confirmed through inspection of raw API responses.

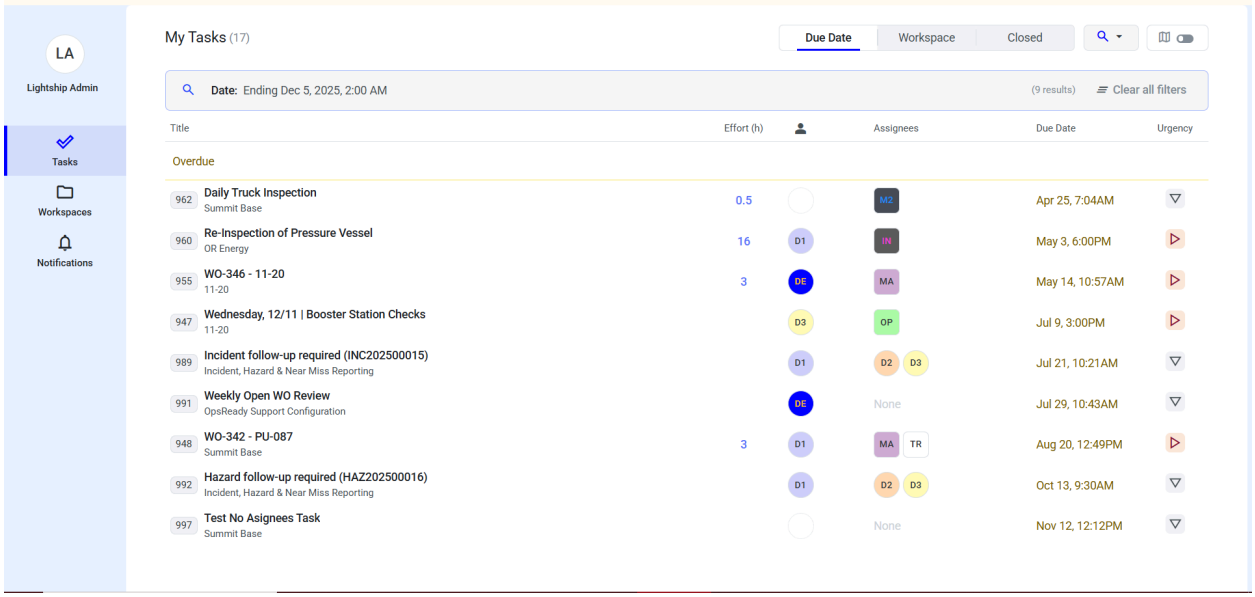
Example of Claude output against OpsReady website showcasing the mentioned pressure vessel inspection:





Example of Claude output against OpsReady website showcasing some of the overdue tasks.





Integration Testing with Claude (MCP)

Integration testing verified that Claude successfully discovered available tools, selected appropriate tools based on natural-language intent, and executed them through the MCP server.

Multi-tool reasoning was observed when Claude combined outputs from multiple tools to generate a single response.
Protocol errors caused by debug logging were identified and resolved to ensure valid JSON responses.

6.3 Error Handling & Edge Case Testing

Edge cases were tested, including users with no assigned tasks, tasks missing due dates or assignees, and invalid input formats.

The system handled these scenarios gracefully by returning informative messages rather than failing or crashing.

Examples:

- Users with no tasks
- Tasks with missing due dates
- Empty task lists
- Invalid input formats (wrong date)

7. Appendixes:

Week 1: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael Smith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris
Summary: The team had a call and discussed some project ideas / concerns that we had. We discussed potentially how we would break up the work and a general overview of the project. We communicated and set up the first meeting with the client, which eventually got cancelled as the client could not make it. The meeting has been rescheduled.
Task Completion from Last Week
● Introduced and created a meeting with the client (meeting got cancelled)
● Group call with the team to discuss project outline Task for This Week ● Meeting with the client for the first meeting on September 16th
● Gather project requirements from client
● Meet with team ideally 2 times after meeting with client to finalize roles and start working on the project
Additional Information
None this week.
Known issues / things blocking progress
The first meeting with the client was initially scheduled for September 12th. Client had to cancel due to a conflict. This made it difficult to get a project overview and begin work.

Week 2: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael Smith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris
Summary: The team had our first call with the client. We met with Michael and Stanley, who are both part of the OpsReady team and the people we will be working with. We determined weekly meeting times that work for all of us for the term and a secondary meeting where we can meet for the first bit of the term if needed. Understood project MVP and got more info from Michael on what exactly is wanted from us and the access that we will be given to acquire their data. Discussed OpsReady data privacy concerns and determined the technologies they would like to use.
Task Completion from Last Week
● Met with the client
● Determined two meetings weekly that work for all of us
● Gathered MVP and requirements for the project Task for This Week
● Meet with Client to go over data access and get a demo site so we can learn the OpsReady tool
● Understand the OpsReady tool
● Potentially get access to sandbox AWS from Michael
● Meeting with group after client call to all learn OpsReady tool together

Additional Information
None this week.
Known issues / things blocking progress
None this week.

Week 3: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael Smith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris
Summary: Met with the client, got access to OpsReady demo site so we can test the data. Got access to Bruno so we can view the current API's that the company uses and if any can help us. Still waiting for AWS access so we can create our own API's. Decided on a fixed weekly meeting for us to meet without the client to do work.
Task Completion from Last Week
- Met with client and got access to Bruno and API files
- Got access to demo site
- Met with internal group after client call to discuss the demo site Task for This Week
- Make sure we get AWS access
- Get a walkthrough on AWS from client
- If we get AWS access, develop simple API to pull some data from demo site
- Ensure internal team's Bruno is working correctly
- Create a diagram to see how everything will work together
- Discuss with internal team on topics for client meeting

Additional Information
None this week.
Known issues / things blocking progress
Need AWS access to continue progress, at a halt right now without it. One teammates Bruno is not working, will get this fixed before client call on Tuesday.

Week 4: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael Smith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris
Summary: Met with the client on Tuesday and discussed aws access, got a demo site tour, and asked questions regarding API's. Also met with the client on thursday for a stack/general overview of the project pitch. Pitched the technologies that we wanted to use, got confirmation from the client and got access to aws. Client requested a few documents detailing the architecture of how the technologies will work, sending them over Monday. Developed some python that authenticates a user that we can use in our project.
Task Completion from Last Week
- Got AWS access
- Got walkthrough of demo site
- Developed piece of code useful to us
- Got internal teams Bruno working

● Developed some diagrams to help us picture the project Task for This Week
● We are using Model Context Protocol (MCP) so our agent can interact with the clients API's, MCP calls tasks as "tools". Need to develop one tool to make sure it can hit the clients site using their api
● Look into AWS bedrock and decide on agent and start fine tuning/training
Additional Information
None this week.
Known issues / things blocking progress
None so far, we are starting to code though so im sure issues will pop up

Week 5: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael SMith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris-Odubo
Summary: Last week we started coding, writing the functions to retrieve user data (TGT, ST tokens). We also finished the get_workspace function, fixed the api session, and added a tool for retrieving users logged in from a specific date. For this week, we will focus on figuring out how we will host the agent using Claude, and developing the Lambda function.
Task Completion from Last Week
● Completed get_tgt, get_st, get_workspace functions
● Fixed the API session so now the get_workspace function works and we can create other functions using this session.
● Added logic for getting users who have logged in since a certain date Task for This Week
● Research Claude and start using it to test functions instead of testing locally on our devices.
● This week we'll be focusing on AWS making sure we have all the permissions we need and developing the custom tool with Lambda and writing the OpenAI schema. We want to be ready to point the tool to the S3 bucket so we can then start creating the bedrock agent.
● By the end of this week, we should:
    ○ Have the basic functions for retrieving specific users and workspaces (which we have done).
    ○ Research Claude and understand how it will be integrated with the agent.
    ○ Developed the Lambda function with the OpenAI schema which should be ready to point to the S3 bucket
Additional Information
- Didn't meet with Micheal and Stan because we really didn't have anything new to discuss.
- We plan to meet on Thursday with Micheal and Stan to show the progress on the Lambda function and/or Claude.
Known issues / things blocking progress
- N/A

Week 6: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael Smith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris

Summary: Met with the client on tuesday and asked a few questions about certain API endpoints, got a good demo and some advice on what endpoints to return specific data. We have been working on tools and getting claude set up for the UI and natural language processing. We got claude connected to our MCP server and have tools that are working, user can ask claude “who was the last person to login?” and claude returns the data. Working on more tools and have a demo on the progress with client on tuesday.
Task Completion from Last Week
● Got Claude working with MCP server
● Got multiple tools working and returning data Task for This Week
● Develop a way to authenticate first, so it does not need to generate new token each query
● Show client a progress update on tuesday
● Develop a react front end (possibly for authentication)
Additional Information
None this week.
Known issues / things blocking progress
None this week

Week 7: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael Smith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris
Summary: Met with the client on Tuesday, scheduled time for midterm review on Thursday Oct 27. Showed them some of our progress and got feedback and more ideas for tools. Got our demo site updated to use their newest APIs, this will allow us to make more tools. Been getting good progress done on tools, assigned tasks to each of us to complete by midterm review.
Task Completion from Last Week
● Showed client demo
● React front end being developed
● Task for This Week
● Work on tools
● Present official midterm review
Additional Information
None this week.
Known issues / things blocking progress
None this week

Week 8: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael Smith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris
Summary: Had a midterm review with the client on Thursday. Prioritizing having plans for the next steps in writing to present to them in the meeting on Tuesday. Also meeting with the developer to discuss how we can access certain APIs that we are struggling with.
Task Completion from Last Week

● Presented midterm review
● React front end being developed
● Task for This Week
● Work on tools
● Discuss concrete next steps with client
Additional Information
None this week.
Known issues / things blocking progress
API access, discussing with developer tomorrow

Week 9: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael Smith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris
Summary: Met with client on Nov 4 discussed next step plans and some more tool ideas. Got the form submissions working so we can now access the data that was submitted by people, this allows us to query more results and get deeper data. We missed our meeting last week on Nov 11 due to the holiday, we spent majority of the week from nov 4 - 11 just trying to get form results working so we did not have a ton of updates anyways. Meeting with the client tomorrow.
Task Completion from Last Week
● Got form submission data working
● Discussed concrete next steps with client
● Finished more tools
● Confirmed with client how to test our results, need to manually compare to what the website says, no other method that they use Task for This Week
● Discuss with client end of project, Solidify if they have any other tools they for sure want to see
● Start working on AWS deployment
Additional Information
None this week.
Known issues / things blocking progress
None this week.

Week 10: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael Smith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris
Summary: Met with the client on Nov 17, discussed documentation, hand off, testing, and final features, they sent us an email highlighting the things they require. The client reached out to us to see if we could present a progress update to their CTO, highlighting how the system and code works. We presented for about half an hour on Nov 21, explaining how the server works and how our code makes the calls and returns the data. We have been working on tools, finishing up major tools before we start to test.
Task Completion from Last Week

● Got clients requirements for end of project, testing, features, documentation, handoff
● Added more tools Task for This Week
● Finish working on all final features of the project
Additional Information
None this week.
Known issues / things blocking progress
None this week.

Week 11: Weekly Report for COMP 4910 Project Course RAG AI Agent for OpsReady Michael Smith Blake Alberding-Berge, Aryan Joglekar, Gesi Morris
Summary: Met with the client and discussed a few issues we were having with workspaces not showing up, got it figured out. Talked to them about possibly doing another presentation for their company. Working on final documentation and presentation and report this week. Not adding many more features. Testing against the site to make sure the agent is getting accurate data.
Task Completion from Last Week
● Finished major features Task for This Week
● Finish testing
● Document all code
● Work on final presentation
Additional Information
None this week.
Known issues / things blocking progress
None this week.

Phase 2: Solo Production Development (Post-course)

Executive Summary

After completing the initial proof-of-concept, I independently rebuilt the OpsReady AI chatbot as a production-ready full-stack web application. This phase focused on creating a user-friendly frontend interface and addressing the scalability challenges that companies face when analyzing their operational data.

The Problem Revisited

Companies using OpsReady's services generate massive amounts of operational data—tasks, work orders, deficiencies, assets, and team activities. While this data is valuable for decision-making, accessing and understanding it remains a significant challenge:

- **Data Overload**: Managers struggle to navigate through large datasets manually
- **Hidden Insights**: Relationships between data points are difficult to identify
- **Time-Consuming Queries**: Finding specific information requires extensive searching
- **Technical Barriers**: Non-technical staff cannot easily query databases

The original solution successfully demonstrated AI-driven data querying through Claude Desktop, but it lacked a critical component: a web-based frontend that managers could access from anywhere.

Phase 2 Objectives

Building on the proof-of-concept, I set out to:

1. **Create a web-based interface** - Make the chatbot accessible through any browser
2. **Maintain data integration** - Preserve all 14 OpsReady API tools developed in Phase 1
3. **Add voice input** - Enable hands-free querying for busy managers
4. **Enable deployment** - Make the application production-ready and scalable
5. **Create demo version** - Build a testable version after losing sandbox access

Architecture Evolution

The system was rebuilt from a local MCP server into a modern three-tier web architecture:

Backend (Python - FastAPI)

- RESTful API with `/api/chat` and `/api/health` endpoints
- Anthropic Claude API integration with tool calling
- Asynchronous request handling
- CORS configuration for secure cross-origin requests
- Environment-based configuration management

Frontend (React + Vite)

- Single-page application with real-time chat interface
- Web Speech API integration for voice input
- Conversation state management
- Responsive design for desktop and mobile
- Auto-scrolling message display

AI Integration Layer

- Claude Sonnet 4 with custom tool definitions

- Agentic loop handling tool calls iteratively
- Conversation history management for context
- Error handling and retry logic

Key Features Implemented

1. Natural Language Interface

- Users can ask questions in plain English
- AI interprets intent and selects appropriate tools
- Responses formatted in clear, readable markdown

2. Voice Input

- Click-to-record microphone button
- Browser-based speech recognition
- Auto-transcription to text input
- Visual recording indicator

3. Multi-Tool Integration All 14 original OpsReady API tools were integrated:

- Task management (overdue, assigned, summaries)
- Work order tracking
- Deficiency monitoring
- Asset inventory
- User activity feeds
- Team coordination
- Login history

4. Production-Ready Features

- Health check endpoint for monitoring
- Error messages with actionable guidance
- Loading states and typing indicators
- Clear chat functionality
- Session persistence

Development Challenges & Solutions

Challenge 1: Frontend Development Timeline

Problem: The original project timeline didn't allow for frontend development. The proof-of-concept worked through Claude Desktop, which isn't accessible to end users.

Solution: After the course ended, I independently built a complete React frontend from scratch, including:

- Chat interface with message history
- Input controls with keyboard shortcuts
- Voice recording with visual feedback
- Responsive CSS with modern animations

Challenge 2: Lost Sandbox Access

Problem: After the project concluded, OpsReady's sandbox credentials expired. I could no longer test the frontend with real data or provide live demos to recruiters.

Solution: Created three separate implementations to maximize demonstration value:

`server.py` - Original MCP Version

- Preserved original team implementation
- Video demo available showing real data integration
- Documents proof-of-concept validation

`api_server.py` - Production Version

- Full OpsReady API integration with CAS authentication
- All 14 tools querying real endpoints
- Cannot demo live but code demonstrates real-world integration

`api_server_mock.py` - Demo Version

- Realistic mock data hardcoded in Python
- No credentials required
- Fully functional live demo
- Same user experience as production

This approach allows me to:

- Show the original working prototype
- Display production-ready code
- Provide interactive live demo

Challenge 3: Authentication Complexity

Problem: OpsReady uses CAS (Central Authentication Service) with TGT/ST ticket flow, requiring careful session management.

Solution:

- Implemented proper authentication flow in `opsready.py`
- Added CSRF token handling for POST requests
- Built retry logic for expired sessions
- Documented authentication patterns for future developers

Conclusion

Phase 2 transformed the OpsReady AI chatbot from a proof-of-concept into a production-ready web application. Despite losing sandbox access, the project demonstrates:

- Full-stack development capability through complete backend and frontend implementation
- Problem-solving skills by creating alternative demonstration methods
- Production readiness with deployment configuration and error handling
- Real-world integration through documented API patterns and authentication flows

The three-version approach (original MCP, production API, mock demo) provides comprehensive evidence of technical ability while maintaining professional standards around client data and credential management.

Repository:
Live Demo: [Vercel URL]
Original Demo Video: [YouTube/Loom URL]